INTRODUCTION PAR L'EXEMPLE A PHP7, ECMASCRIPT6, VUE.JS et NUXT.JS

Serge Tahé, septembre 2020

Ce document réunit en un unique document une série de quatre articles publiés en 2019 :

1. |Introduction au langage PHP7 par l'exemple| ;
2. |Introduction au langage ECMASCRIPT 6 par l'exemple| ;
3. |Introduction au framework VUE.JS par l'exemple| ;
4. |Introduction au framework NUXT.JS par l'exemple| ;

Ce sont tous des documents pour **débutants**. Les articles ont une suite logique mais **sont faiblement couplés** :

- le document **[1]** présente le langage PHP 7. Le lecteur seulement intéressé par le langage PHP et pas par le langage Javascript des articles suivants s'arrêtera là ;
- les documents **[2-4]** visent à construire un client Javascript au serveur de calcul de l'impôt développé dans le document **[1]** ;
- les frameworks Javascript **[vue.js]** et **[nuxt.js]** des articles 3 et 4 nécessitent de connaître le Javascript des dernières versions d'ECMASCRIPT, celles de la version 6. Le document **[2]** est donc destiné à ceux qui ne connaissent pas cette version de Javascript. Il fait référence au serveur de calcul de l'impôt construit dans le document **[1]**. Le lecteur de **[2]** aura alors parfois besoin de se référer au document **[1]** ;
- une fois ECMASCRIPT 6 maîtrisé, on peut aborder le framework VUE.JS qui permet de construire des clients Javascript s'exécutant dans un navigateur en mode SPA (Single Page Application). C'est le document **[3]**. Il fait référence à la fois au serveur de calcul de l'impôt construit dans le document **[1]** et au code du client Javascript autonome construit en **[2]**. Le lecteur de **[3]** aura alors parfois besoin de se référer aux documents **[1]** et **[2]** ;
- une fois VUE.JS maîtrisé, on peut aborder le framework NUXT.JS qui permet de construire des clients Javascript s'exécutant dans un navigateur en mode SSR (Server Side Rendered). Il fait référence à la fois au serveur de calcul de l'impôt construit dans le document **[1]**, au code du client Javascript autonome construit en **[2]** ainsi qu'à l'application **[vue.js]** développée dans le document **[3]**. Le lecteur de **[4]** aura alors parfois besoin de se référer aux documents **[1]** **[2]** et **[3]** ;

Les codes des quatre articles sont disponibles |ici|.

Serge Tahé, septembre 2020

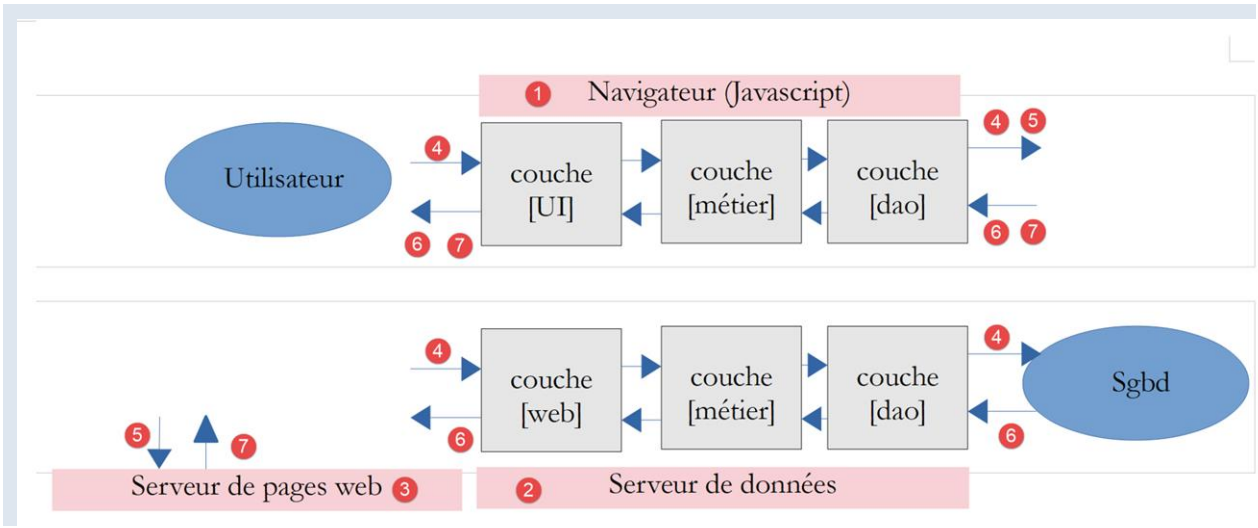
Table des matières

1 Introduction au langage PHP7 par l'exemple

1.1 Présentation du cours

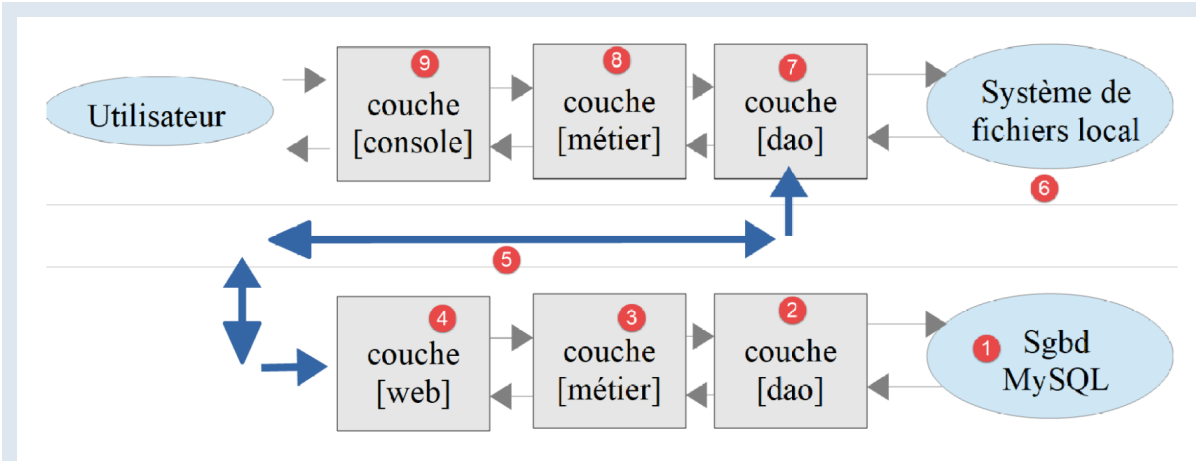
Ce chapitre propose une liste de **scripts console** PHP 7 dans différents domaines (structures du langage, accès aux fichiers, aux bases de données, au réseau internet). La programmation web est abordée via des **services web**. On a appelé dans ce document, service web, toute application web produisant du texte brut. Ce sont des serveurs de données et non des serveurs de pages web qui sont un mixte HTML, CSS et Javascript. On y aborde des concepts web classiques (protocole HTTP, réponses jSON ou XML, gestion de session, authentification) également utilisés dans la programmation web classique.

De nos jours, il est fréquent de construire des applications web en mode client / serveur :



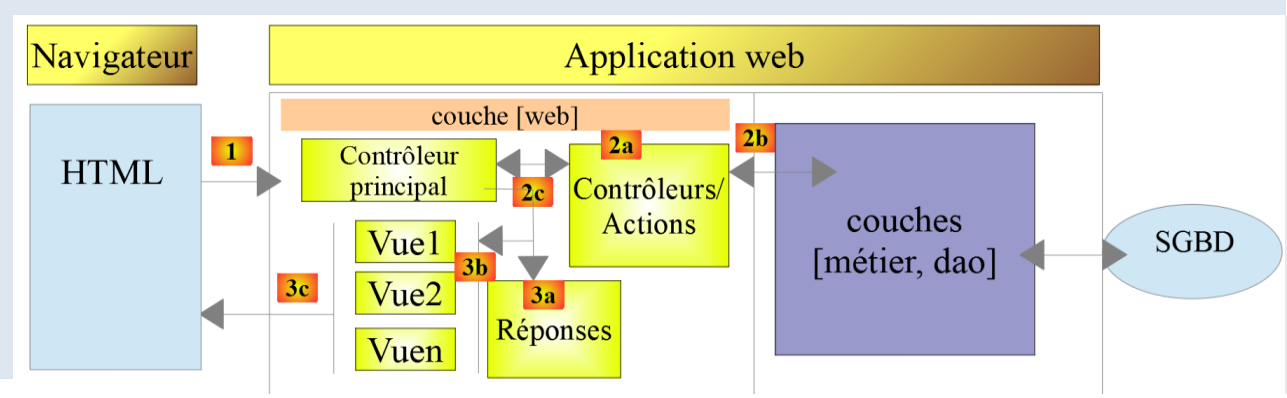
- en **[1]**, le navigateur web affiche des pages web à destination d'un utilisateur **[5, 7]**. Ces pages contiennent du Javascript implémentant un client d'un service web de données **[2]** ainsi qu'un client d'un serveur de fragments de pages web **[3]**. Un framework JS bien établi dans ce domaine est **Angular 2** de Google (mai 2019) ;
- en **[2]**, le serveur web est un serveur de données. Il peut être écrit dans n'importe quel langage. Il ne produit pas de pages web au sens classique (HTML, CSS, Javascript) sauf peut-être la 1ère fois. Mais cette 1ère page peut être obtenue d'un serveur web classique **[3]** (pas un serveur de données). Le Javascript de la page initiale va alors générer les différentes pages web de l'application en obtenant les données **[4]** à afficher auprès du serveur web qui agit comme un serveur de données **[2]**. Il peut également obtenir des fragments de page web **[5]** pour habiller ces données auprès du serveur de pages web **[3]** ;
- en **[4]**, l'utilisateur initie une action ;
- en **[6,7]** : il reçoit des données habillées par un fragment de page web ;

Nous allons dans ce document écrire des applications client / serveur en PHP7 ayant la structure suivante :



On a là une application client / serveur écrite en PHP. Un script console **[9]** interrogera un serveur de données **[4]**. Ce qui sera appris ici pour écrire le service de données pourra être réutilisé dans une application web. Le service de données en PHP pourra être conservé et le client PHP sera lui remplacé par un client Javascript.

Comme fil rouge du document, nous construirons un service de calcul de l'impôt en 13 versions. La version 13 aura l'architecture suivante :



La couche **[web]** du serveur aura une architecture MVC (Model – View – Controller). Tout le cours PHP 7 vise à construire cette version.

Les scripts de ce document sont commentés et leur exécution console reproduite. Des explications supplémentaires sont parfois fournies. Le document nécessite une lecture active : pour comprendre un script, il faut à la fois lire son code, ses commentaires et ses résultats d'exécution.

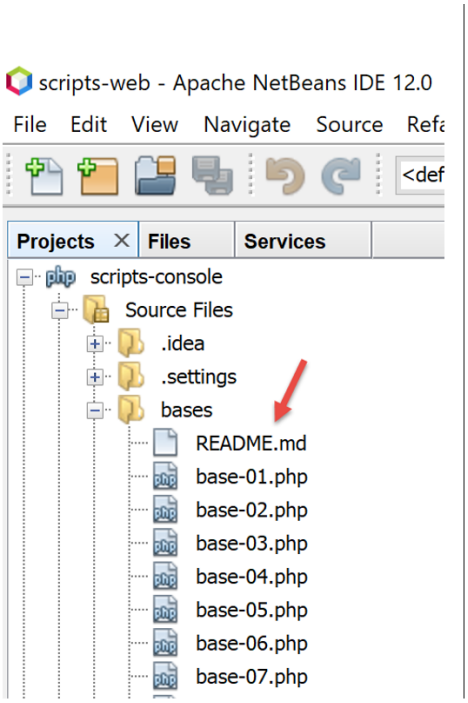
Les exemples du document sont disponibles |**ici**|.
L'application serveur PHP 7 peut être testée |**ici**|.

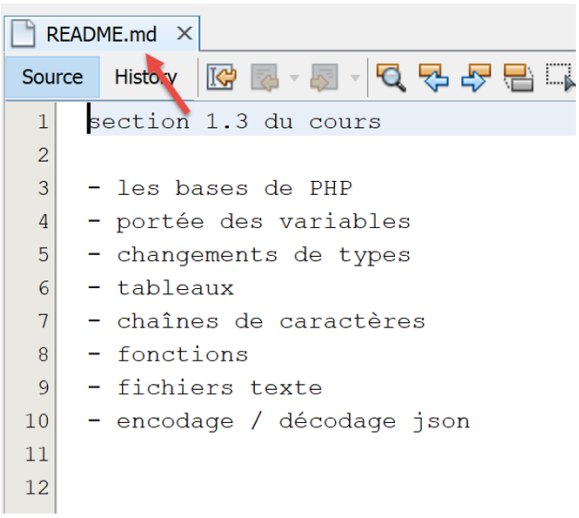
Serge Tahé, juillet 2019

Le contenu du cours est le suivant :

Chapitre 10	Applications en couches : application en couches et programmation par interfaces, exemple d'une architecture à trois couches
Chapitre 11	Version 4 de l'exercice d'application : restructuration de la version 3 avec une architecture trois couches, tests des couches avec le framework Codeception
Chapitre 12	Utilisation du SGBD MySQL : création d'une base de données, création d'une table, remplissage d'une table, exécution d'ordres SQL, ordres SQL préparés, utilisation de transactions
Chapitre 13	Version 5 de l'exercice d'application : implémentation avec une architecture trois couches et une base de données MySQL
Chapitre 14	Version 6 de l'exercice d'application : installation du SGBD PostgreSQL, implémentation avec une architecture trois couches et une base de données PostgreSQL
Chapitre 15	Version 7 de l'exercice d'application : regroupement des paramètres de l'application dans un fichier jSON
Chapitre 16	Fonctions internet : découverte des protocoles HTTP, SMTP, POP3 et IMAP et implémentation de ces protocoles par programme
Chapitre 17	Services web : page statique / page dynamique PHP, le langage HTML, service web jSON de date / heure, environnement d'un service web, passage de paramètres à un service web, sessions web, authentification
Chapitre 18	Version 8 de l'exercice d'application : application client / serveur, architecture trois couches, écriture client et serveur, tests Codeception
Chapitre 19	Version 9 de l'exercice d'application : application client / serveur, architecture trois couches, écriture client et serveur, tests Codeception, utilitaires **`[Logger]`** **`[SendAdminMail]`**
Chapitre 20	Version 10 de l'exercice d'application : application client / serveur, utilisation d'un serveur Redis côté serveur pour mémoriser des informations de portée Application
Chapitre 21	Traitement de documents XML
Chapitre 22	Version 11 de l'exercice d'application : application client / serveur, le client et le serveur s'échangent du XML
Chapitre 23	Version 12 de l'exercice d'application : application client / serveur, par rapport aux versions précédentes où le client et le serveur échangeaient soit du jSON soit du XML, la version 12 permet d'échanger indifféremment du jSON, du XML ou du HTML, la version HTML suit l'architecture MVC (Modèle-Vue-Contrôleur), écriture du serveur, définition des actions autorisées par le serveur, contrôleur principal et contrôleurs secondaires, types de réponse jSON, XML, HTML du serveur, les vues de l'application HTML, écriture d'un client jSON
Chapitre 24	Version 13 de l'exercice d'application : sécurisation de l'application web

Certains lecteurs préfèreront lire, modifier et tester le code plutôt que de lire un cours. Dans chaque dossier des codes de ce document on a mis un fichier **`[README.md]`** résumant le contenu du dossier et faisant le lien avec le cours :


scripts-web - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refa
Projects
Files
Services
scripts-console
Source Files
.idea
.settings
bases
README.md
base-01.php
base-02.php
base-03.php
base-04.php
base-05.php
base-06.php
base-07.php


README.md
Source
History
1 section 1.3 du cours
2
3 - les bases de PHP
4 - portée des variables
5 - changements de types
6 - tableaux
7 - chaînes de caractères
8 - fonctions
9 - fichiers texte
10 - encodage / décodage json
11
12

1.2 Installation d'un environnement de travail

Les scripts ont été écrits et testés dans l'environnement suivant :

- un environnement serveur web Apache / SGBD MySQL / PHP 7.3 appelé **Laragon ;**
- l'IDE de développement **Netbeans 10.0 ;**

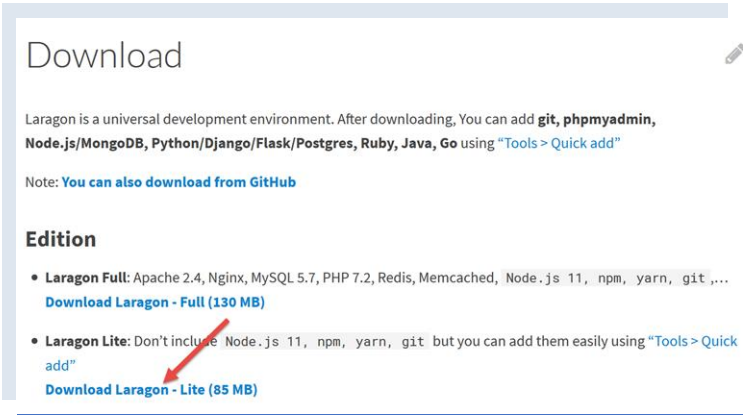
1.2.1 Installation de Laragon

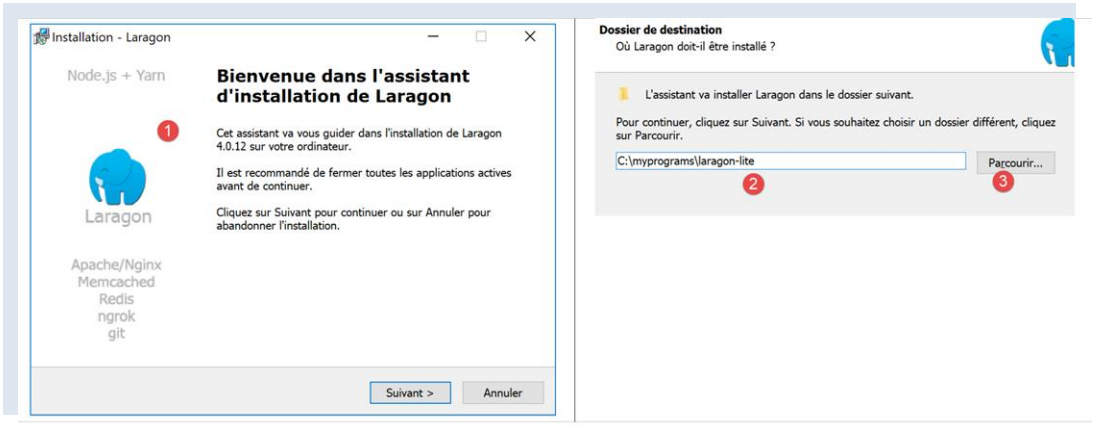
Laragon est un package réunissant plusieurs logiciels :

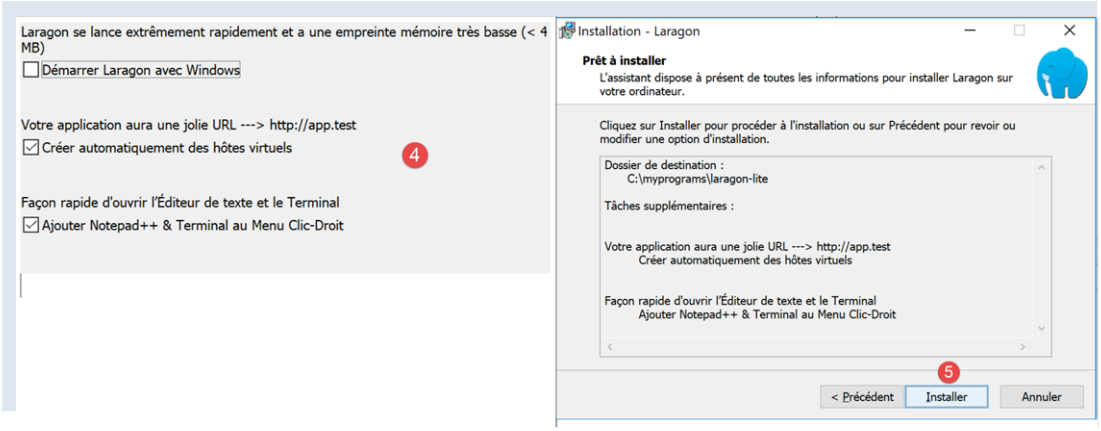
- un serveur web Apache. Nous l'utiliserons pour l'écriture de scripts web en PHP ;
- le SGBD MySQL ;
- le langage de script PHP ;
- un serveur Redis implémentant un cache pour des applications web :

Laragon peut être téléchargé (mars 2019) à l'adresse suivante :

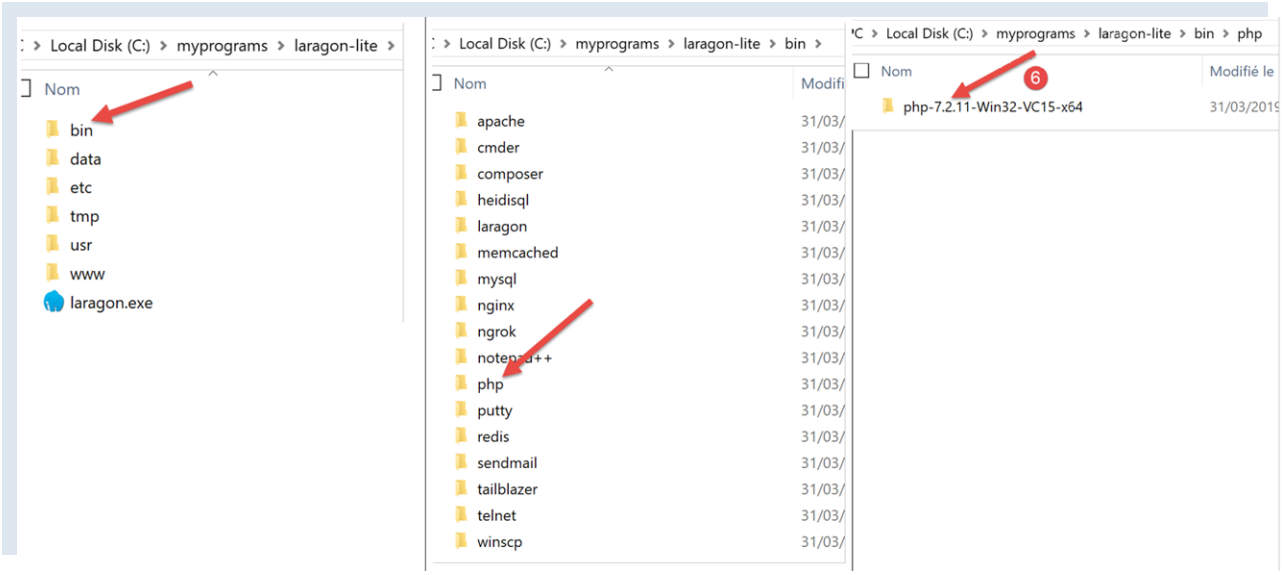
https://laragon.org/download/





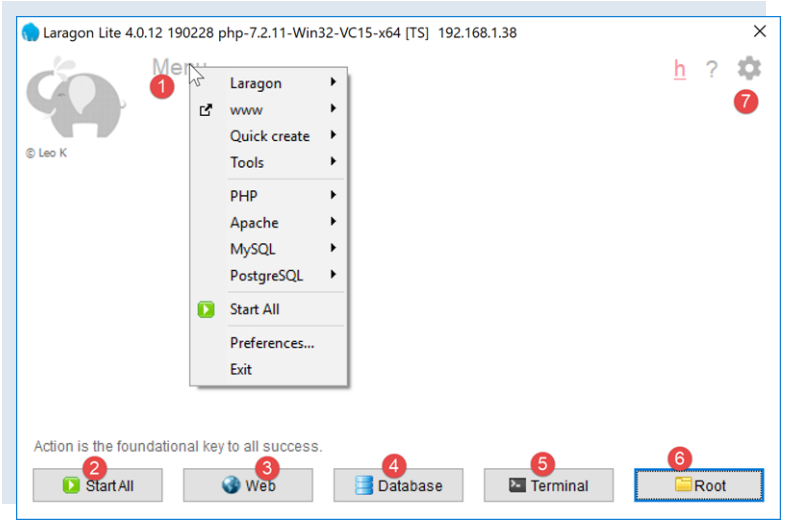


- l'installation **[1-5]** donne naissance à l'arborescence suivante :



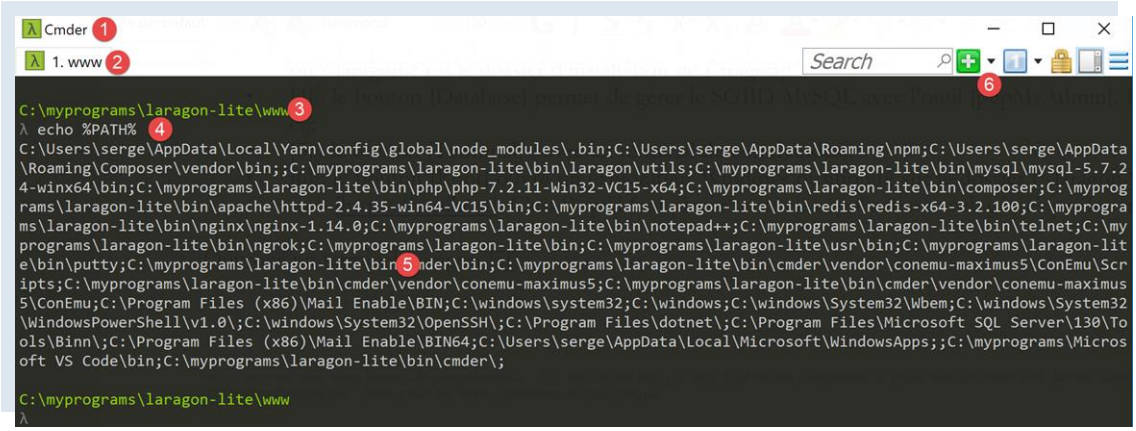
- en **[6]** le dossier d'installation de PHP ;

Le lancement de **[Laragon]** affiche la fenêtre suivante :



- **[1]** : le menu principal de Laragon ;
- **[2]** : le bouton **[Start All]** lance le serveur web Apache et le SGBD MySQL ;
- **[3]** : le bouton **[WEB]** affiche la page web **[http://localhost]** qui correspond au fichier PHP **[<laragon>/www/index.php]** où <laragon> est le dossier d'installation de Laragon ;
- **[4]** : le bouton **[Database]** permet de gérer le SGBD MySQL avec l'outil **[phpMyAdmin]**. Il faut auparavant installer celui-ci ;
- **[5]** : le bouton **[Terminal]** ouvre un terminal de commandes ;
- **[6]** : le bouton **[Root]** ouvre un explorateur Windows positionné sur le dossier **[<laragon>/www]** qui est la racine du site web **[http://localhost]**. C'est là qu'il faut placer toutes les applications web gérées par le serveur Apache de Laragon ;

Ouvrons un terminal Laragon **[5]** :

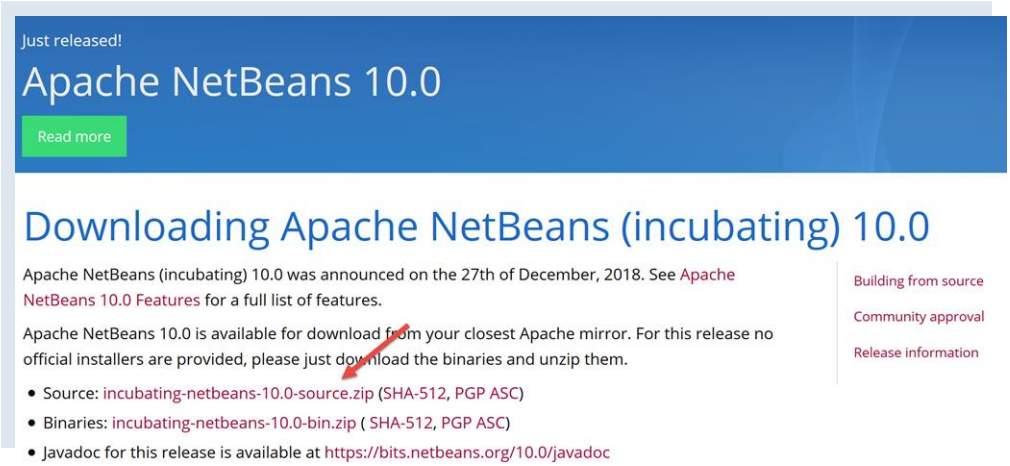


- en **[1]**, le type du terminal. Trois types de terminaux sont disponibles en **[6]** ;
- en **[2, 3]** : le dossier courant ;
- en **[4]**, on tape la commande **[echo %PATH%]** qui affiche la liste des dossiers explorés lors de la recherche d'un exécutable. Tous les principaux dossiers de Laragon sont inclus dans ce chemin des exécutables, ce qui ne serait pas le cas si on ouvrait une fenêtre de commandes **[cmd]** dans Windows. Dans ce document, lorsqu'on est amené à taper des commandes pour installer tel ou tel logiciel, c'est en général dans un terminal Laragon que ces commandes sont tapées ;

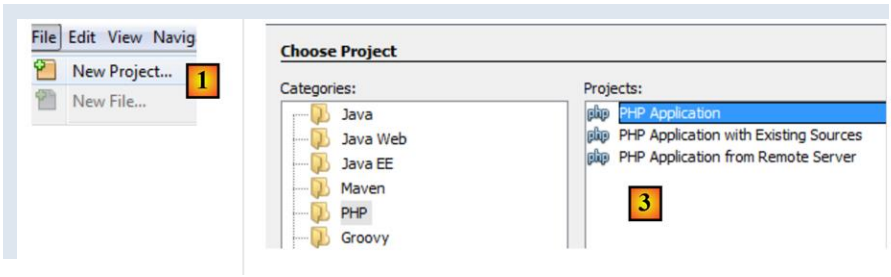
1.2.2 Installation de l'IDE Netbeans 10.0

L'IDE Netbeans 10.0 peut être téléchargé à l'adresse suivante (mars 2019) :

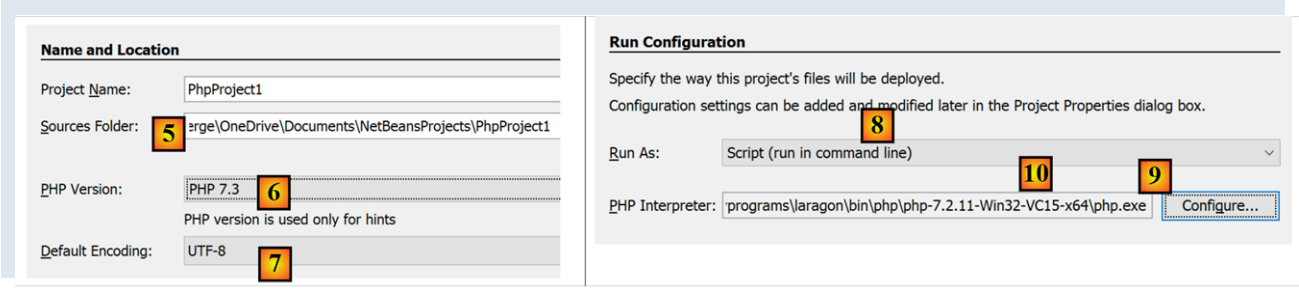
|https://netbeans.apache.org/download/index.HTML|



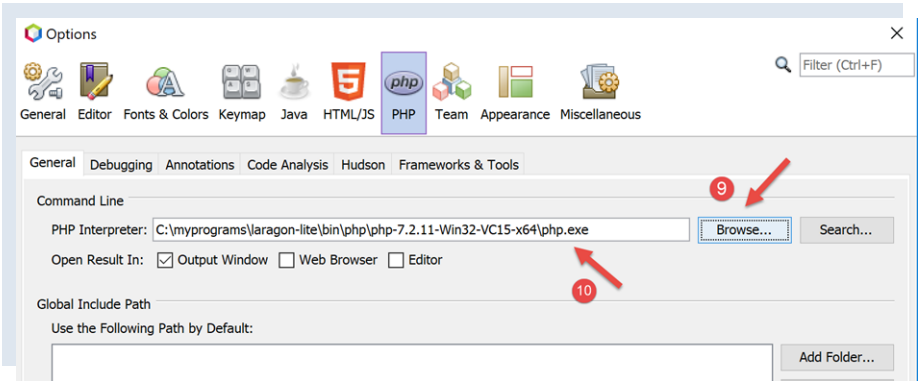
Le fichier téléchargé est un zip qu'il suffit de dézipper. Une fois Netbeans installé et lancé, on peut créer un premier projet PHP.



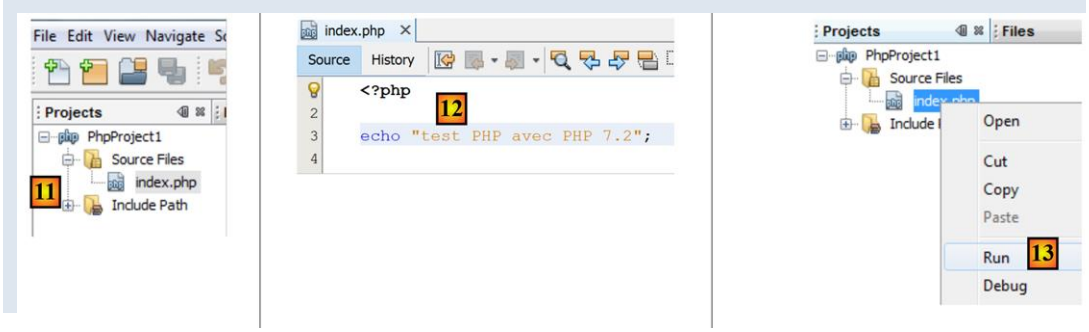
- en **[1]**, prendre l'option File / New Project ;
- en **[2]**, prendre la catégorie **[PHP]** ;
- en **[3]**, prendre le type de projet **[PHP Application]** ;



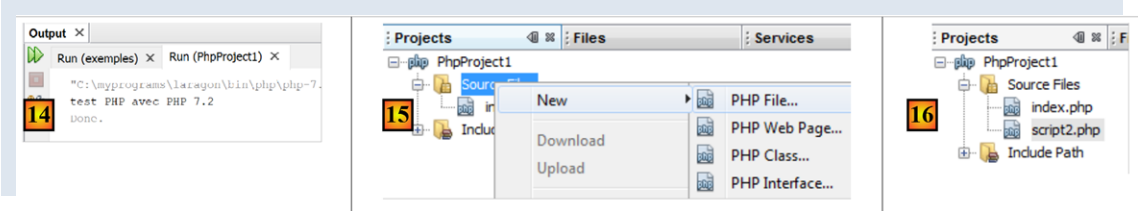
- en **[4]**, donner un nom au projet ;
- en **[5]**, choisir un dossier pour le projet ;
- en **[6]**, choisir la version de PHP téléchargée ;
- en **[7]**, choisir l'encodage UTF-8 pour les fichiers PHP ;
- en **[8]**, choisir le mode **[Script]** pour exécuter les scripts PHP en mode ligne de commande. Choisir **[Local WEB Server]** pour exécuter un script PHP dans un environnement web ;
- en **[9,10]**, indiquer le répertoire d'installation de l'interpréteur PHP du package Laragon :



- choisir **[Finish]** pour terminer l'assistant de création du projet PHP ;

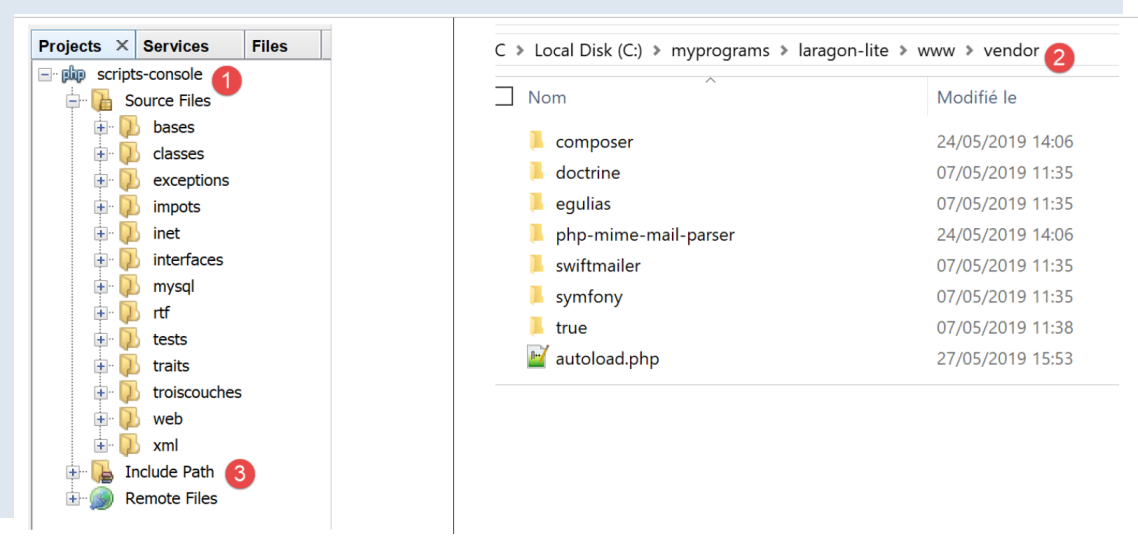


- en **[11]**, le projet est créé avec un script **[index.php]** ;
- en **[12]**, on écrit un script PHP minimal ;
- en **[13]**, on exécute **[index.php]** ;

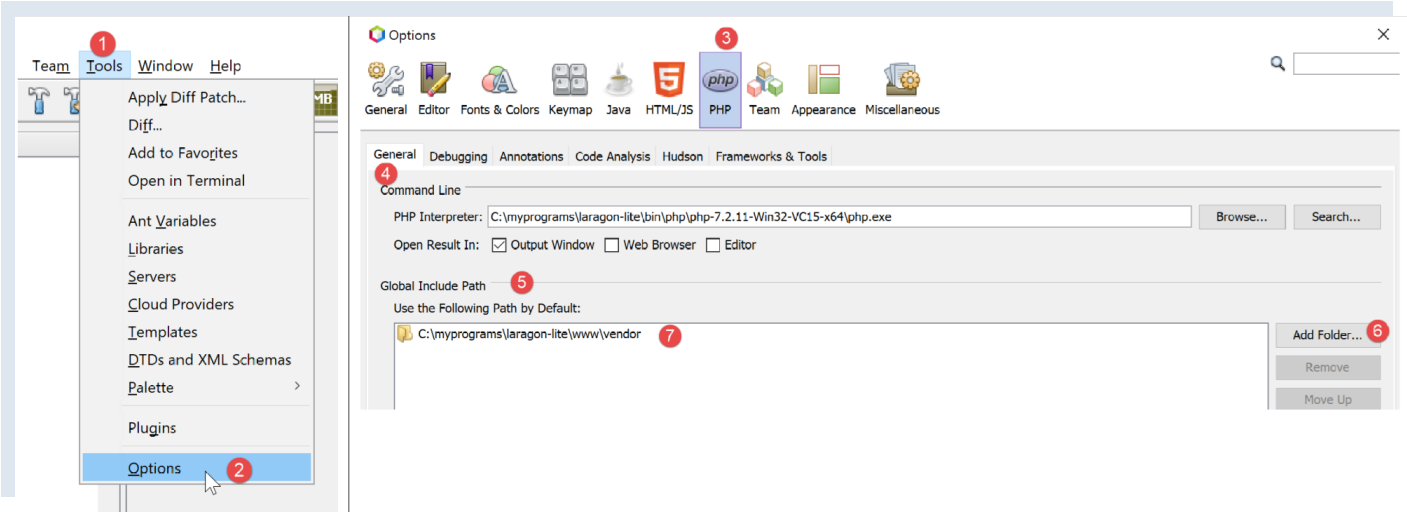


- en **[14]**, les résultats dans la fenêtre **[output]** de Netbeans ;
- en **[15]**, on crée un nouveau script ;
- en **[16]**, le nouveau script ;

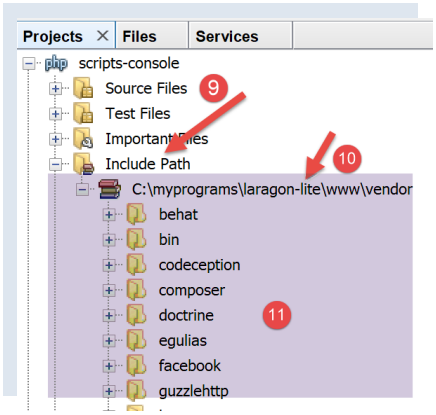
Le lecteur pourra créer tous les scripts qui vont suivre dans différents dossiers du même projet PHP. Les codes source des scripts de ce document sont disponibles sous la forme de l'arborescence Netbeans suivante :



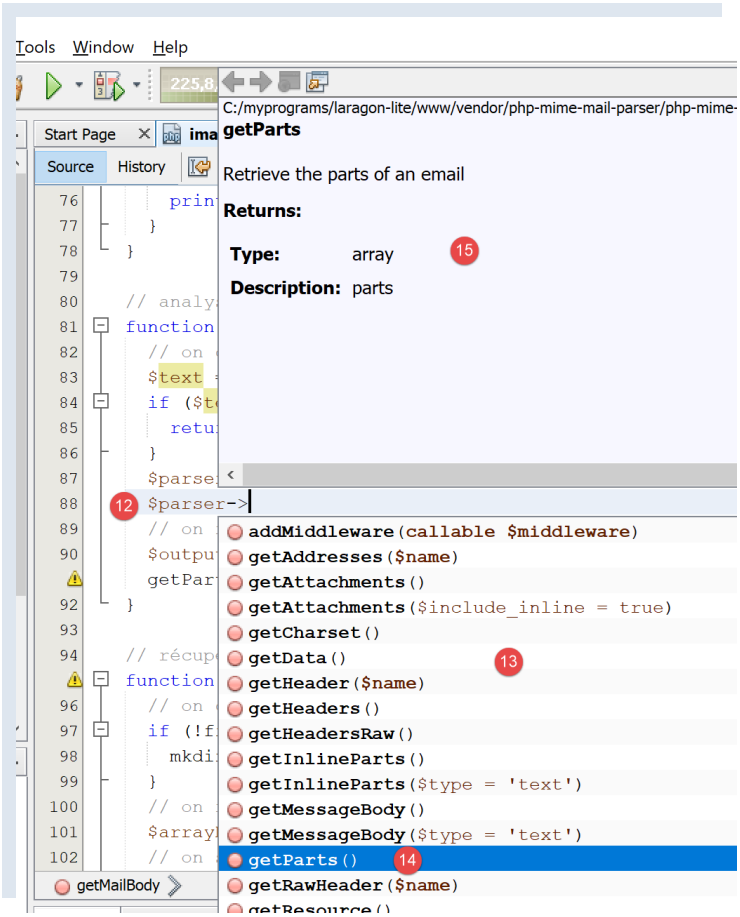
Les scripts de ce document sont placés dans l'arborescence du projet **[scripts-console] [1]**. Nous allons utiliser également des bibliothèques PHP qui seront placées dans le dossier **[<laragon-lite>/www/vendor] [2]** où <laragon-lite> est le dossier d'installation du logiciel Laragon. Pour que Netbeans reconnaisse les bibliothèques de **[2]** comme faisant partie du projet **[scripts-console]**, il nous faut inclure le dossier **[vendor] [2]** dans la branche **[Include Path] [3]** du projet. Nous allons configurer Netbeans pour que le dossier **[<laragon-lite>/www/vendor] [2]** soit inclus dans tout nouveau projet PHP et pas seulement dans le projet **[scripts-console]** :



- en **[1-2]**, on va dans les options de Netbeans ;
- en **[3-4]**, on configure les options de PHP ;
- en **[5-7]**, on configure le **[Global Include Path]** de PHP : les dossiers indiqués en **[7]** sont automatiquement inclus dans le **[Include Path]** de tout projet PHP ;



- en **[9]**, on accède aux propriétés de la branche **[Include Path]** ;
- en **[10-11]**, les nouvelles bibliothèques explorées par Netbeans. Netbeans explore le code PHP de ces bibliothèques et mémorise leurs classes, interfaces, fonctions… afin de pouvoir proposer de l'aide au développeur ;



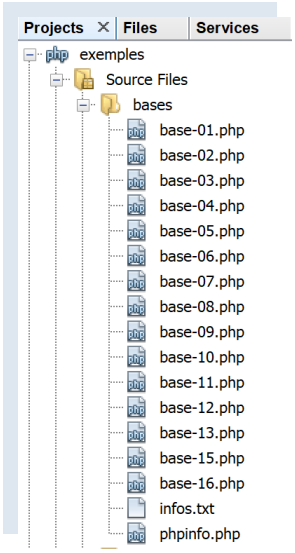
- en **[12]**, un code utilise la classe **[PhpMimeMailParser\Parser]** de la bibliothèque **[vendor/php-mime-mail-parser]** ;
- en **[13]**, Netbeans propose les méthodes de cette classe ;
- en **[14-15]**, Netbeans affiche la documentation de la méthode sélectionnée ;

La notion d'**[Include Path]** est ici propre à Netbeans. PHP a également cette notion mais ce sont a priori deux notions différentes.

Maintenant que l'environnement de travail a été installé, nous pouvons aborder les bases de PHP.

1.3 Les bases de PHP

1.3.1 L'arborescence des scripts



1.3.2 Configuration de PHP

PHP arrive préconfiguré par un fichier texte **[php.ini]**. Toutes ces configurations peuvent être changées par programmation. La configuration de PHP influence grandement l'exécution des scripts. Il est donc important de la connaître. Le script suivant **[phpinfo.php]** le permet :

```
1    <?php
2
3    phpinfo();
```

Commentaires

- ligne 3 : la fonction **[phpinfo]** affiche la configuration de PHP ;

Résultats de l'exécution

```
1   "C:\myprograms\laragon-lite\bin\php\php-7.2.11-Win32-VC15-x64\php.exe" "C:\Data\st-2019\dev\php7\php5-
    exemples\exemples\tests\phpinfo.php"
2   phpinfo()
3   PHP Version => 7.2.11
4
5   System => Windows NT DESKTOP-528I5CU 10.0 build 17134 (Windows 10) AMD64
6   Build Date => Oct 10 2018 01:57:32
7   Compiler => MSVC15 (Visual C++ 2017)
8   Architecture => x64
9   Configure Command => cscript /nologo configure.js  "--enable-snapshot-build" "--enable-debug-pack" "--with-
    pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-
    build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-
    dotnet=shared" "--without-analyzer" "--with-pgo"
10  Server API => Command Line Interface
11  Virtual Directory Support => enabled
12  Configuration File (php.ini) Path => C:\windows
13  Loaded Configuration File => C:\myprograms\laragon-lite\bin\php\php-7.2.11-Win32-VC15-x64\php.ini
14  Scan this dir for additional .ini files => (none)
15  Additional .ini files parsed => (none)
16  …
17  Done.
```

La fonction **[phpinfo]** affiche ici plus de 800 lignes de configuration. Nous n'allons pas les commenter car la plupart concerne un usage avancé de PHP. Une ligne importante est la ligne 13 ci-dessus : elle indique quel fichier **[php.ini]** a été utilisé pour configurer le PHP que vous allez utiliser pour exécuter vos scripts. Si vous souhaitez modifier la configuration d'exécution de PHP, c'est ce

fichier qu'il vous faut modifier. De nombreux commentaires sont présents dans ce fichier pour expliquer le rôle des différentes configurations.

1.3.3 Un premier exemple

1.3.3.1 Le code

Ci-dessous, on trouvera un programme **[bases-01.php]** présentant les premières caractéristiques de PHP.

```
1   <?php
2
3   // ceci est un commentaire
4   // variable utilisée sans avoir été déclarée
5   $nom = "dupont";
6   // un affichage écran
7   print "nom=$nom\n";
8   // un tableau avec des éléments de types différents
9   $tableau = array("un","deux",3,4);
10  // son nombre d'éléments
11  $n = count($tableau);
12  // une boucle
13  for ($i = 0; $i < $n; $i ++) {
14    print "tableau[$i]=$tableau[$i]\n";
15  }
16  // initialisation de 2 variables avec le contenu d'un tableau
17  list ($chaine1, $chaine2) = array("chaine1","chaine2");
18  // concaténation des 2 chaînes
19  $chaine3 = $chaine1 . $chaine2;
20  // affichage résultat
21  print "[$chaine1,$chaine2,$chaine3]\n";
22  // utilisation fonction
23  affiche($chaine1);
24  // le type d'une variable peut être connu
25  afficheType("n", $n);
26  afficheType("chaine1", $chaine1);
27  afficheType("tableau", $tableau);
28  // le type d'une variable peut changer en cours d'exécution
29  $n = "a changé";
30  afficheType("n", $n);
31  // une fonction peut rendre un résultat
32  $res1 = f1(4);
33  print "res1=$res1\n";
34  // une fonction peut rendre un tableau de valeurs
35  list ($res1, $res2, $res3) = f2();
36  print "(res1,res2,res3)=[$res1,$res2,$res3]\n";
37  // on aurait pu récupérer ces valeurs dans un tableau
38  $t = f2();
39  for ($i = 0; $i < count($t); $i ++) {
40    print "t[$i]=$t[$i]\n";
41  }
42  // des tests
43  for ($i = 0; $i < count($t); $i ++) {
44    // n'affiche que les chaînes
45    if (getType($t[$i]) === "string") {
46      print "t[$i]=$t[$i]\n";
47    }
48  }
49  // opérateurs de comparaison == et ===
50  if ("2" == 2) {
51    print "avec l'opérateur ==, la chaîne 2 est égale à l'entier 2\n";
52  } else {
53    print "avec l'opérateur ==, la chaîne 2 n'est pas égale à l'entier 2\n";
54  }
55  if ("2" === 2) {
56    print "avec l'opérateur ===, la chaîne 2 est égale à l'entier 2\n";
57  } else {
58    print "avec l'opérateur ===, la chaîne 2 n'est pas égale à l'entier 2\n";
59  }
60  // d'autres tests
61  for ($i = 0; $i < count($t); $i ++) {
62    // n'affiche que les entiers >10
63    if (getType($t[$i]) === "integer" and $t[$i] > 10) {
64      print "t[$i]=$t[$i]\n";
65    }
66  }
67  // une boucle while
```

```
68  // initialisation tableau (PHP 7)
69  $t = [8,5,0,- 2,3,4];
70  $i = 0;
71  $somme = 0;
72  while ($i < count($t) and $t[$i] > 0) {
73    print "t[$i]=$t[$i]\n";
74    $somme += $t[$i]; // $somme=$somme+$t[$i]
75    $i ++; // $i=$i+1
76  } // while
77  print "somme=$somme\n";
78
79  // notation d'une constante
80  const TAUX_TVA = 19.6;
81  print "taux de TVA=" . TAUX_TVA;
82
83  // arrêt du programme
84  exit();
85
86  // affiche
87  // ----------------------------------
88  function affiche($chaine) {
89    // affiche $chaine
90    print "chaine=$chaine\n";
91  }
92
93  // afficheType
94  // ----------------------------------
95  function afficheType($name, $variable) {
96    // affiche le type de $variable
97    print "type[variable $" . $name . "]=" . getType($variable) . "\n";
98  }
99
100 // ----------------------------------
101 function f1($param) {
102   // ajoute 10 à $param
103   return $param + 10;
104 }
105
106 // ----------------------------------
107 function f2() {
108   // rend 3 valeurs
109   return array("un",0,100);
110 }
```

Les résultats :

```
1.  nom=dupont
2.  tableau[0]=un
3.  tableau[1]=deux
4.  tableau[2]=3
5.  tableau[3]=4
6.  [chaine1,chaine2,chaine1chaine2]
7.  chaine=chaine1
8.  type[variable $n]=integer
9.  type[variable $chaine1]=string
10. type[variable $tableau]=array
11. type[variable $n]=string
12. res1=14
13. (res1,res2,res3)=[un,0,100]
14. t[0]=un
15. t[1]=0
16. t[2]=100
17. t[0]=un
18. avec l'opérateur ==, la chaîne 2 est égale à l'entier 2
19. avec l'opérateur ===, la chaîne 2 n'est pas égale à l'entier 2
20. t[2]=100
21. t[0]=8
22. t[1]=5
23. somme=13
24. taux de TVA=19.6
```

Commentaires

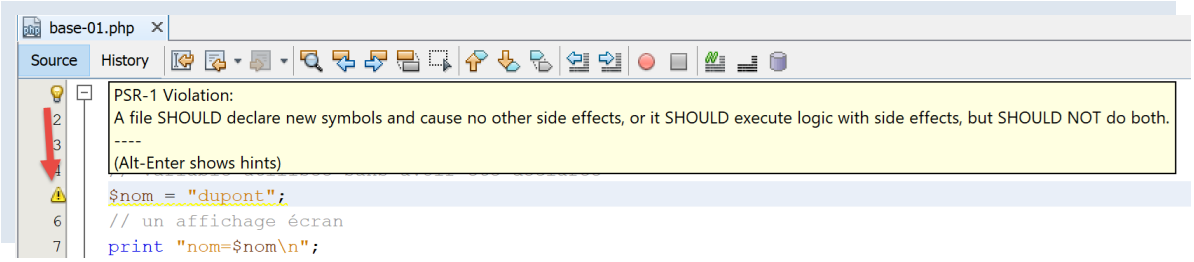
- ligne 5 : en PHP, on ne déclare pas le type des variables. Celles-ci ont un type dynamique qui peut varier au cours du temps. $**nom** représente la variable d'identifiant *nom ;*

- ligne 7 : pour écrire à l'écran, on peut utiliser l'instruction *print* ou l'instruction *echo ;*
- ligne 9 : le mot clé *array* permet de définir un tableau. La variable *$nom**[$i]*** représente l'élément *$i* du tableau *$tableau ;*
- ligne 11 : la fonction *count($tableau)* rend le nombre d'éléments du tableau *$tableau* ;
- lignes 13-15 : une boucle. Celle-ci n'ayant qu'une instruction, les accolades sont alors facultatives. Dans la suite de ce document, nous mettrons systématiquement les accolades quelque soit le nombre d'instructions ;
- ligne 14 : les chaînes de caractères sont entourées de guillemets " ou d'apostrophes '. A l'intérieur de guillemets, les variables $*variable* sont évaluées mais pas à l'intérieur d'apostrophes ;
- ligne 17 : la fonction *list* permet de rassembler des variables dans une liste et de leur attribuer une valeur avec une unique opération d'affectation. Ici *$chaine1="chaine1"* et *$chaine2="chaine2*" ;
- ligne 19 : l'opérateur . est l'opérateur de concaténation de chaînes ;
- lignes 83-86 : le mot clé *function* définit une fonction. Une fonction rend ou non des valeurs par l'instruction *return*. Le code appelant peut ignorer ou récupérer les résultats d'une fonction. Une fonction peut être définie n'importe où dans le code.
- ligne 92 : la fonction prédéfinie *getType($variable)* rend une chaîne de caractères représentant le type de *$variable*. Ce type peut changer au cours du temps ;
- ligne 45 : l'opérateur === compare deux éléments de façon stricte : il faut qu'ils aient le **même type** pour être comparés. L'opérateur == est moins strict : deux éléments peuvent être égaux **sans être du même type**. C'est ce que montrent les instructions des lignes 50-59. Dans le cas de l'opérateur ==, la comparaison se fait après transtypage des deux éléments comparés dans un même type. Des conversions implicites ont alors lieu. Il est assez facile « d'oublier » la présence de ces conversions implicites et d'aboutir ainsi à des résultats imprévus, tels que de découvrir qu'une condition est vraie alors que vous l'attendiez fausse. Pour éviter cet écueil, nous utiliserons systématiquement l'opérateur de compraison === ;
- ligne 63 : on peut utiliser également les opérateurs booléens **or** et **! ;**
- ligne 69 : à la place de la notation **array()**, on peut utiliser la notation **[]** pour initialiser un tableau depuis PHP7 ;
- ligne 84 : la fonction prédéfinie *exit* arrête l'exécution du script ;
- ligne 110 : la balise **?>** signale la fin du script PHP. Elle n'est pas indispensable. De plus, dans un contexte web, elle peut poser problème si elle est suivie par des espaces ou des marques de fin de ligne, difficiles à détecter car non visibles dans un éditeur de texte. Aussi dans la suite du document, nous omettrons systématiquement cette balise ;

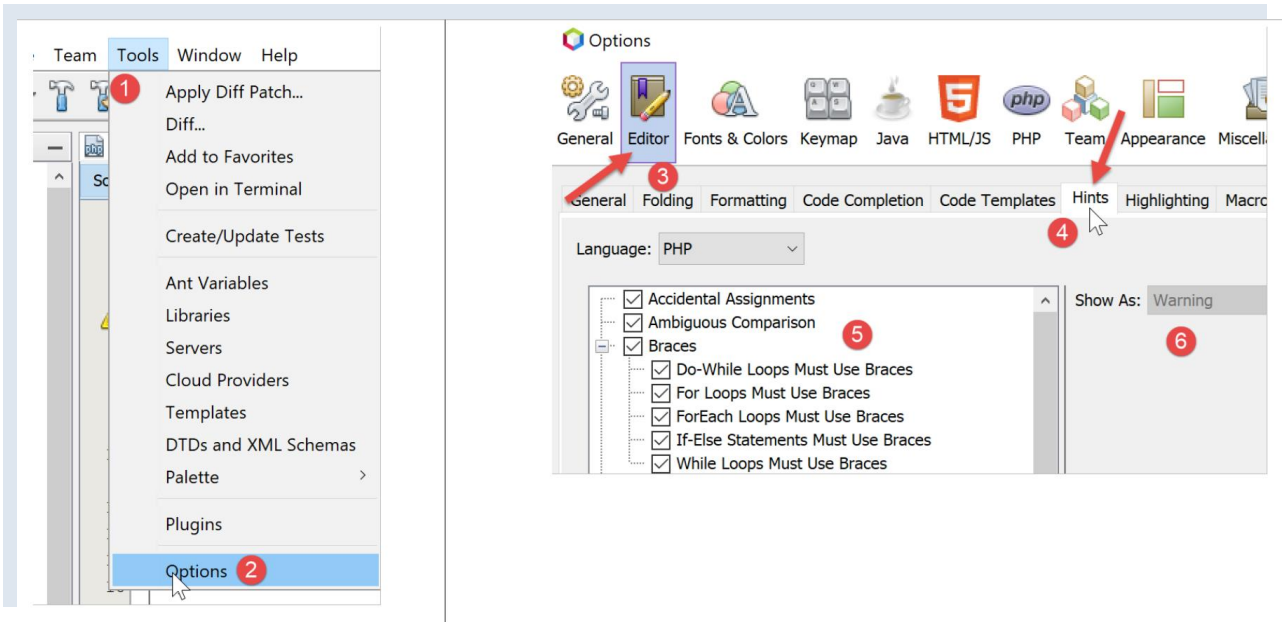
Note : dans ce document, nous utiliserons le mot clé **[print]** pour afficher du texte sur la console. Une autre méthode pour faire la même chose est d'utiliser le mot clé **[echo]**. Il y a de subtiles différences entre ces deux mots clés, mais dans le contexte de ce document il n'y en aura aucune. Si donc vous préférez utiliser **[echo]**, alors faites-le.

1.3.3.2 Usage de Netbeans

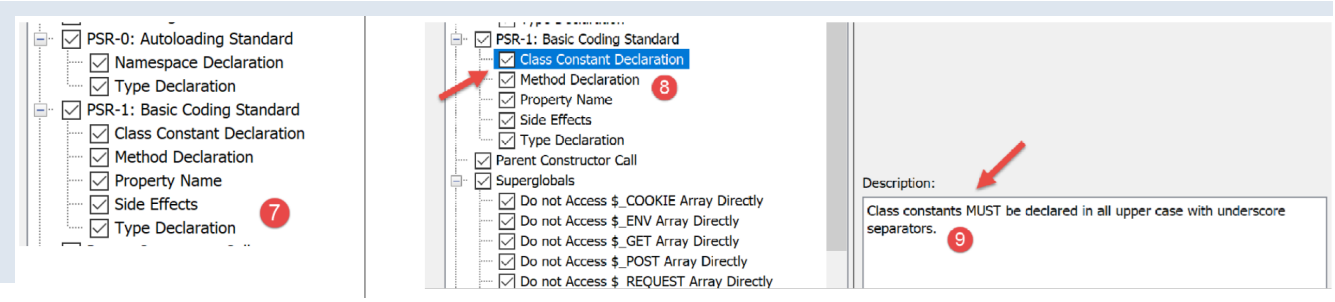
Netbeans émet divers avertissements qu'il est utile de vérifier. Prenons l'exemple du script **[bases-01.php]** :



A la ligne 5, Netbeans émet un avertissement comme quoi le fichier ne respecte pas la recommandation PSR-1 (PHP Standard Recommendations n° 1). Les PSR sont des recommandations pour produire un code standard et ainsi faciliter l'interopérabilité et la maintenance de codes écrits par différentes personnes. Il peut être ennuyeux d'avoir des avertissements si on veut transgresser délibérément les standards, parce que par exemple l'équipe du projet en a d'autres. Ce que l'on souhaite vérifier ou pas avec Netbeans est configurable :



- en **[5]**, on trouve les éléments que l'on veut contrôler avec Netbeans ;
- en **[6]**, le niveau de sévérité assigné à l'erreur signalée par Netbeans ;



On voit en **[7]** qu'on a demandé à contrôler que le code suit bien les recommandations PSR-0 et PSR-1. Il n'y a rien d'obligatoire. En phase d'apprentissage du langage, ll est conseillé de contrôler le maximum d'options proposées par Netbeans. On apprend ainsi beaucoup de choses. On adaptera ensuite ce contrôle de Netbeans aux normes de codage de l'équipe d'un projet.

Voyons les normes de codage PSR-1 **[8, 9]** :

	Option	Contrôle
1	**Class Constant Declaration**	Class constants MUST be declared in all upper case with underscore separators. Ex : *const TAUX_TVA*
2	**Method Declaration**	Method names MUST be declared in camelCase(). Ex : *public function executeBatchImpots*{}
3	**Property Name**	Property names SHOULD be declared in $StudlyCaps, $camelCase, or $under_score format (consistently in a scope) Ex : *public SalaireAnnuel* (StudlyCaps), *public salaireAnnuel* (camelCase), *public salaire_annuel* (under_score)
4	**Side Effects**	A file SHOULD declare new symbols and cause no other side effects, or it SHOULD execute logic with side effects, but SHOULD NOT do both.
5	**Type Declaration**	Type names MUST be declared in StudlyCaps (Code written for 5.2.x and before SHOULD use the pseudonamespacing convention of Vendor_ prefixes on type names). Each type is in a file by itself, and is in a namespace of at least one level: a top-level vendor name. Ex : *class EtudiantBoursier* {}

La recommandation PSR-1 / 4 dit que dans un fichier PHP, on doit trouver :

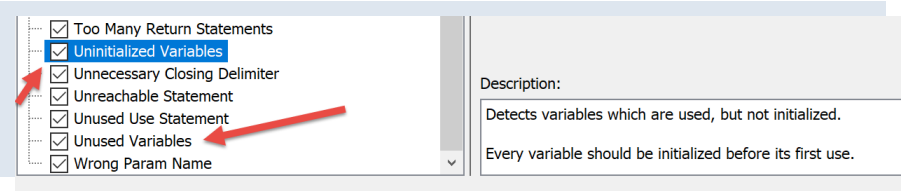
- soit la déclaration d'un type (classes, interface) ;

- soit du code exécutable sans déclaration de nouveaux types ;

Il existe d'autres recommandations PHP non contrôlées par Netbeans : PSR-3, PSR-4, PSR-6, PSR-7 et PSR-13.

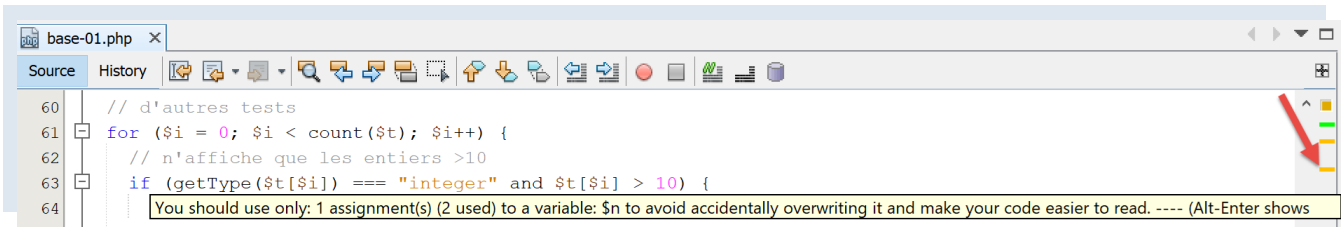
Par facilité, les exemples du document ne vérifient pas tous la recommandation PSR-1 car cela oblige à dispatcher le code des classes et interfaces dans des fichiers séparés, ce qui est trop lourd pour des exemples basiques. Il est alors plus facile de tout mettre dans un fichier. Pour l'exemple de l'application présentée comme fil rouge de ce document, on a cherché à observer au mieux la recommandation PSR-1.

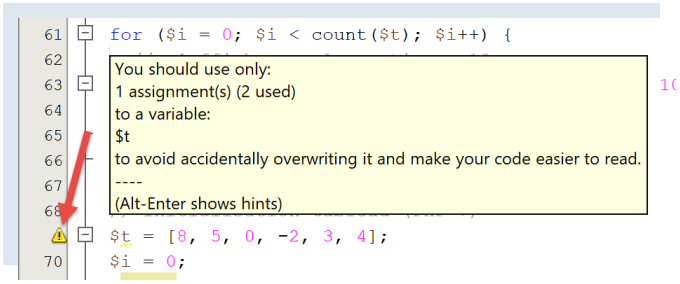
Certains avertissements de Netbeans signalent une erreur potentielle :



L'avertissement **`[Unitialized Variables]`** signale une erreur probable, souvent une erreur de frappe sur le nom d'une variable. Il en est de même pour l'avertissement **`[Unused Variables]`**.

Finalement, il est conseillé de vérifier tous les avertissements de Netbeans signalés par un panneau dans la marge gauche du code et un tiret jaune dans la marge droite :





1.3.4 La portée des variables

1.3.4.1 Exemple 1

Le script **`[bases-02.php]`** est le suivant :

```
1   <?php
2
3   // portée des variables
4   function f1() {
5     // on utilise la variable globale $i
6     global $i;
7     $i ++;
8     $j = 10;
9     print "f1[i,j]=[$i,$j]\n";
10  }
11
12  function f2() {
13    // on utilise la variable globale $i
14    global $i;
```

```
15    $i ++;
16    $j = 20;
17    print "f2[i,j]=[$i,$j]\n";
18  }
19
20  function f3() {
21    // on utilise une variable locale $i
22    $i = 4;
23    $j = 30;
24    print "f3[i,j]=[$i,$j]\n";
25  }
26
27  // tests
28  $i = 0;
29  $j = 0; // ces deux variables ne sont connues d'une fonction f
30          // que si celle-ci déclare explicitement par l'instruction global
31          // qu'elle veut les utiliser
32  f1();
33  f2();
34  f3();
35  print "test[i,j]=[$i,$j]\n";
```

Les résultats :

```
1   f1[i,j]=[1,10]
2   f2[i,j]=[2,20]
3   f3[i,j]=[4,30]
4   test[i,j]=[2,0]
```

Commentaires

- lignes 28-29 : définissent 2 variables *$i* et *$j* du programme principal. Ces variables ne sont pas connues à l'intérieur des fonctions. Ainsi, ligne 9, la variable *$j* de la fonction *f1* est une variable **locale** à la fonction *f1* et est différente de la variable *$j* du programme principal. Une fonction peut accéder à une variable *$variable* du programme principal via le mot clé **global ;**
- ligne 7, l'instruction désigne la variable globale *$i* du programme principal ;

1.3.4.2 Exemple 3

Le script **[bases-03.php]** est le suivant :

```
1   <?php
2
3   // la portée d'une variable est globale aux blocs de code
4   $i = 0;
5   {
6     $i = 4;
7     $i++;
8   }
9   print "i=$i\n";
```

Les résultats :

```
1    i=5
```

Commentaires

Dans certains langages, une variable définie à l'intérieur d'accolades a la portée de celles-ci : elle n'est pas connue à l'extérieur de celles-ci. Les résultats ci-dessus montrent qu'il n'en est rien en PHP. La variable $i définie ligne 5 à l'intérieur des accolades est la même que celle utilisée lignes 4 et 8 à l'extérieur de celles-ci.

1.3.5 Les changements de types

Les variables en PHP n'ont pas un type constant. Celui-ci peut changer en cours d'exécution selon la valeur affectée à la variable. Dans des opérations impliquant des données de divers types, l'interpréteur PHP fait des conversions implicites pour ramener les opérandes dans un type commun. Ces conversions implicites, si elles ne sont pas connues du développeur, peuvent être une source d'erreurs difficiles à repérer. On présente ci-dessous, un script **[bases-04.php]** montrant des conversions implicites et explicites :

```
1.  <?php
2.
3.  // types stricts dans le passage des paramètres
4.  declare(strict_types=1);
5.
```

```
6.  // changements implicites de types
7.  // type -->bool
8.  print "Conversion vers un booléen-----------------------------\n";
9.  showBool("abcd", "abcd");
10. showBool("", "");
11. showBool("[1, 2, 3]", [1, 2, 3]);
12. showBool("[]", []);
13. showBool("NULL", NULL);
14. showBool("0.0", 0.0);
15. showBool("0", 0);
16. showBool("4.6", 4.6);
17.
18. function showBool(string $prefixe, $var): void {
19.   print "(bool) $prefixe : ";
20.   // la conversion de $var en booléen se fait automatiquement dans le test qui suit
21.   if ($var) {
22.     print "true";
23.   } else {
24.     print "false";
25.   }
26.   print "\n";
27. }
28.
29. // changements implicites de type string vers un type numérique
30. // string --> nombre
31. print "Conversion chaîne vers nombre-----------------------------\n";
32. showNumber("12");
33. showNumber("45.67");
34. showNumber("abcd");
35.
36. function showNumber(string $var): void {
37.   $nombre = $var + 1;
38.   var_dump($nombre);
39.   print "($var): $nombre\n";
40. }
41.
42. // changements explicites de type
43. // vers int
44. showInt("12.45");
45. showInt(67.8);
46. showInt(TRUE);
47. showInt(NULL);
48.
49. function showInt($var): void {
50.   print "paramètre : ";
51.   var_dump($var);
52.   print "\n";
53.   print "résultat de la conversion : ";
54.   var_dump((int) $var);
55.   print "\n";
56. }
57.
58. // vers float
59. showFloat("12.45");
60. showFloat(67);
61. showFloat(TRUE);
62. showFloat(NULL);
63.
64. function showFloat($var): void {
65.   print "paramètre : ";
66.   var_dump($var);
67.   print "\n";
68.   print "résultat de la conversion : ";
69.   var_dump((float) $var);
70.   print "\n";
71. }
72.
73. // vers string
74. showstring(5);
75. showString(6.7);
76. showString(FALSE);
77. showString(NULL);
78.
79. function showString($var): void {
80.   print "paramètre : ";
81.   var_dump($var);
82.   print "\n";
```

```
83.   print "résultat de la conversion : ";
84.   var_dump((string) $var);
85.   print "\n";
86. }
```

Commentaires

- ligne 4 : demande une vérification stricte du type des paramètres d'une fonction lorsque celui-ci est précisé ;
- ligne 18 : la fonction **[showBool]** a pour but de montrer la transformation implicite (automatique) que fait l'interpréteur PHP lorsqu'une valeur de tout type doit être transformée en booléen (ligne 21) ;
- ligne 18 : la paramètre $var n'a pas de type assigné. Le paramètre effectif pourra donc être de tout type. Le paramètre $*prefixe* lui devra être de type **string**. La fonction *showBool* ne rend aucune valeur (**void**) ;
- ligne 21 : dans l'instruction **if($var)**, la valeur de $var doit être transformée en booléen pour que le *if* soit évalué. De façon étonnante l'interpréteur PHP a une réponse pour tout type de valeur qu'on lui donne ;
- lignes 9-16 : la valeur du paramètre de la fonction **[showBool]** sera successivement :
  - ligne 9 : une chaîne non vide : résultat TRUE (la casse n'est pas importante, TRUE=true) ;
  - ligne 10 : une chaîne vide : résultat FALSE ;
  - ligne 11 : un tableau non vide : résultat TRUE ;
  - ligne 12 : un tableau vide : résultat FALSE ;
  - ligne 14 : le nombre réel 0 : résultat FALSE ;
  - ligne 15 : le nombre entier 0 : résultat FALSE ;
  - ligne 16 : le nombre réel (ou entier) différent de 0 : résultat TRUE ;

C'est ce que montrent les affichages écrans obtenus :

```
1   Conversion vers un booléen------------------------------
2   (bool) abcd : true
3   (bool)  : false
4   (bool) [1, 2, 3] : true
5   (bool) [] : false
6   (bool) NULL : false
7   (bool) 0.0 : false
8   (bool) 0 : false
9   (bool) 4.6 : true
```

Continuons le code du script :

```
1.  // changements implicites de type string vers un type numérique
2.  // string --> nombre
3.  print "Conversion chaîne vers nombre------------------------------\n";
4.  showNumber("12");
5.  showNumber("45.67");
6.  showNumber("abcd");
7.
8.  function showNumber(string $var): void {
9.    $nombre = $var + 1;
10.   var_dump($nombre);
11.   print "($var): $nombre\n";
12. }
```

Commentaires

- ligne 8 : la fonction **[showNumber]** admet un paramètre de type **string** et ne rend pas de résultat (**void**) ;
- ligne 9 : ce paramètre est utilisé dans une opération arithmétique, ce qui va forcer l'interpréteur PHP à essayer de transformer $var en nombre ;
  - ligne 4 : va transformer la chaîne "12" en nombre entier 12 ;
  - ligne 5 : va transformer la chaîne "45.67" en nombre réel 45.67 ;
  - ligne 6 : va émettre un avertissement mais va quand même transformer la chaîne "abcd" en nombre 0 ;

Voici les résultats de l'exécution :

```
1   Conversion chaîne vers nombre------------------------------
2   int(13)
3   (12): 13
4   float(46.67)
5   (45.67): 46.67
6
7   Warning: A non-numeric value encountered in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_031.php
    on line 37
8   int(1)
```

```
9   (abcd): 1
```

Continuons le code du script :

```
1.  // changements explicites de type
2.  // vers int
3.  showInt("12.45");
4.  showInt(67.8);
5.  showInt(TRUE);
6.  showInt(NULL);
7.
8.  function showInt($var): void {
9.    print "paramètre : ";
10.   var_dump($var);
11.   print "\n";
12.   print "résultat de la conversion : ";
13.   var_dump((int) $var);
14.   print "\n";
15. }
```

Commentaires

- ligne 8 : la fonction **[showInt]** reçoit un paramètre de n'importe quel type et ne rend pas de résultat. Elle essaie de convertir le paramètre $*var* en entier ligne 13. De façon générale, pour changer une variable $*var* en un type T, on écrit **(T) $var** où **T** peut être : **int, integer, bool, boolean, float, double, real, string, array, object, unset** ;
- ligne 3 : convertit la chaîne "12.45" en entier 12 ;
- ligne 4 : convertit le réel 67.8 en entier 67 ;
- ligne 5 : convertit le booléen TRUE en l'entier 1 (le booléen FALSE en l'entier 0) ;
- ligne 6 : convertit le pointeur NULL en entier 0 ;

C'est ce que montrent les affichages écran :

```
1   paramètre : string(5) "12.45"
2
3   résultat de la conversion : int(12)
4
5   paramètre : float(67.8)
6
7   résultat de la conversion : int(67)
8
9   paramètre : bool(true)
10
11  résultat de la conversion : int(1)
12
13  paramètre : NULL
14
15  résultat de la conversion : int(0)
```

Nous continuons l'étude du script avec la conversion explicite de valeurs en type **float** :

```
1.  // vers float
2.  showFloat("12.45");
3.  showFloat(67);
4.  showFloat(TRUE);
5.  showFloat(NULL);
6.
7.  function showFloat($var): void {
8.    print "paramètre : ";
9.    var_dump($var);
10.   print "\n";
11.   print "résultat de la conversion : ";
12.   var_dump((float) $var);
13.   print "\n";
14. }
```

Commentaires

- ligne 7 : la fonction **[showFloat]** reçoit un paramètre de type quelconque et ne rend pas de résultat ;
- ligne 12 : la valeur de ce paramètre est transformée explicitement en *float* ;
- ligne 2 : la chaîne "12.45" est transformée en le nombre réel 12.45 ;
- ligne 3 : l'entier 67 est transformée en le nombre réel 67 ;
- ligne 4 : le booléen TRUE est transformé en le nombre réel 1 (la valeur FALSE en le nombre 0) ;

- ligne 5 : le pointeur NULL est transformé en le nombre réel 0 ;

C'est ce montrent les résultats écran :

```
1   paramètre : string(5) "12.45"
2
3   résultat de la conversion : float(12.45)
4
5   paramètre : int(67)
6
7   résultat de la conversion : float(67)
8
9   paramètre : bool(true)
10
11  résultat de la conversion : float(1)
12
13  paramètre : NULL
14
15  résultat de la conversion : float(0)
```

Nous continuons la présentation du script en étudiant des conversions vers le type **string** :

```
1   // vers string
2   showstring(5);
3   showString(6.7);
4   showString(FALSE);
5   showString(NULL);
6
7   function showString($var) : void {
8     print "paramètre : ";
9     var_dump($var);
10    print "\n";
11    print "résultat de la conversion : ";
12    var_dump((string) $var);
13    print "\n";
14  }
```

- ligne 7 : la fonction **[showString]** reçoit un paramètre de type quelconque et ne rend pas de résultat ;
- ligne 12 : la valeur du paramètre est transformée en un type **string** ;
- ligne 2 : l'entier 5 sera transformé en la chaîne « 5 » ;
- ligne 3 : le réel 6.7 sera transformé en la chaîne « 6.7 » ;
- ligne 4 : le booléen FALSE sera ransformé en chaîne vide ;
- ligne 5 : le pointeur NULL sera transformé en chaîne vide ;

Voici les résultats écran :

```
1   paramètre : int(5)
2
3   résultat de la conversion : string(1) "5"
4
5   paramètre : float(6.7)
6
7   résultat de la conversion : string(3) "6.7"
8
9   paramètre : bool(false)
10
11  résultat de la conversion : string(0) ""
12
13  paramètre : NULL
14
15  résultat de la conversion : string(0) ""
```

1.3.6 Les tableaux

1.3.6.1 Tableaux classiques à une dimension

Le script **[bases-05.php]** est le suivant :

```
1.  <?php
2.
3.  // tableaux classiques
4.  // initialisation
5.  $tab1 = array(0, 1, 2, 3, 4, 5);
```

```php
// parcours - 1
print "tab1 a " . count($tab1) . " éléments\n";
for ($i = 0; $i < count($tab1); $i++) {
  print "tab1[$i]=$tab1[$i]\n";
}
// parcours - 2
print "tab1 a " . count($tab1) . " éléments\n";
reset($tab1);
while (list($clé, $valeur) = each($tab1)) {
  print "tab1[$clé]=$valeur\n";
}
// ajout d'éléments
$tab1[] = $i++;
$tab1[] = $i++;
// parcours - 3
print "tab1 a " . count($tab1) . " éléments\n";
$i = 0;
foreach ($tab1 as $élément) {
  print "tab1[$i]=$élément\n";
  $i++;
}
// suppression dernier élément
array_pop($tab1);
// parcours - 4
print "tab1 a " . count($tab1) . " éléments\n";
for ($i = 0; $i < count($tab1); $i++) {
  print "tab1[$i]=$tab1[$i]\n";
}
// suppression premier élément
array_shift($tab1);
// parcours - 5
print "tab1 a " . count($tab1) . " éléments\n";
for ($i = 0; $i < count($tab1); $i++) {
  print "tab1[$i]=$tab1[$i]\n";
}
// ajout en fin de tableau
array_push($tab1, -2);
// parcours - 6
print "tab1 a " . count($tab1) . " éléments\n";
for ($i = 0; $i < count($tab1); $i++) {
  print "tab1[$i]=$tab1[$i]\n";
}
// ajout en début de tableau
array_unshift($tab1, -1);
// parcours - 7
print "tab1 a " . count($tab1) . " éléments\n";
for ($i = 0; $i < count($tab1); $i++) {
  print "tab1[$i]=$tab1[$i]\n";
}
1
```

Les résultats :

```
1   tab1 a 6 éléments
2   tab1[0]=0
3   tab1[1]=1
4   tab1[2]=2
5   tab1[3]=3
6   tab1[4]=4
7   tab1[5]=5
8   tab1 a 6 éléments
9
10  Deprecated: The each() function is deprecated. This message will be suppressed on further calls in
    C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_04.php on line 14
11  tab1[0]=0
12  tab1[1]=1
13  tab1[2]=2
14  tab1[3]=3
15  tab1[4]=4
16  tab1[5]=5
17  tab1 a 8 éléments
18  tab1[0]=0
19  tab1[1]=1
20  tab1[2]=2
21  tab1[3]=3
22  tab1[4]=4
23  tab1[5]=5
```

```
24  tab1[6]=6
25  tab1[7]=7
26  tab1 a 7 éléments
27  tab1[0]=0
28  tab1[1]=1
29  tab1[2]=2
30  tab1[3]=3
31  tab1[4]=4
32  tab1[5]=5
33  tab1[6]=6
34  tab1 a 6 éléments
35  tab1[0]=1
36  tab1[1]=2
37  tab1[2]=3
38  tab1[3]=4
39  tab1[4]=5
40  tab1[5]=6
41  tab1 a 7 éléments
42  tab1[0]=1
43  tab1[1]=2
44  tab1[2]=3
45  tab1[3]=4
46  tab1[4]=5
47  tab1[5]=6
48  tab1[6]=-2
49  tab1 a 8 éléments
50  tab1[0]=-1
51  tab1[1]=1
52  tab1[2]=2
53  tab1[3]=3
54  tab1[4]=4
55  tab1[5]=5
56  tab1[6]=6
57  tab1[7]=-2
```

Commentaires

Le programme ci-dessus montre des opérations de manipulation d'un tableau de valeurs. Il existe deux notations pour les tableaux en PHP :

```
1   $tableau=array("un",2,"trois")
2   $contraires=array("petit"=>"grand", "beau"=>"laid", "cher"=>"bon marché")
```

Le tableau 1 est appelé **tableau** et le tableau 2 un **dictionnaire** ou **tableau associatif** où les éléments sont notés **clé => valeur**. La notation *$contraires**["beau"]*** désigne la valeur associée à la clé "beau". C'est donc ici la chaîne "laid". Le tableau 1 n'est qu'une variante du dictionnaire et pourrait être noté :

```
$tableau=array(0=>"un",1=>2,2=>"trois")
```

On a ainsi *$tableau**[2]**="trois"*. Finalement, il n'y a que des dictionnaires. Dans le cas d'un tableau classique de *n* éléments, les clés sont les nombres entiers de l'intervalle **[0,n-1]**.

- ligne 14 : la fonction *each($tableau)* permet de parcourir un dictionnaire. A chaque appel, elle rend une paire (clé,valeur) de celui-ci. Comme le montre la ligne 10 des résultats, la fonction **each** est désormais **obsolète** dans PHP 7 ;
- ligne 13 : la fonction *reset($dictionnaire)* positionne la fonction *each* sur la première paire (clé,valeur) du dictionnaire.
- ligne 14 : la boucle *while* s'arrête lorsque la fonction *each* rend une paire vide à la fin du dictionnaire. C'est une conversion implicite qui agit ici : la paire vide est convertie en le booléen FALSE ;
- ligne 18 : la notation *$tableau**[]**=valeur* ajoute l'élément *valeur* comme dernier élément de *$tableau ;*
- ligne 23 : le tableau est parcouru avec un *foreach*. Cet élément syntaxique permet de parcourir un dictionnaire, donc un tableau, selon deux syntaxes :

```
foreach($dictionnaire as $clé=>$valeur)
foreach($tableau as $valeur)
```

  La première syntaxe ramène une paire (*clé,valeur*) à chaque itération alors que la seconde syntaxe ne ramène que l'élément *valeur* du dictionnaire.
- ligne 28 : la fonction *array_pop($tableau)* supprime le dernier élément de $*tableau ;*
- ligne 35 : la fonction *array_shift($tableau)* supprime le premier élément de $*tableau ;*
- ligne 42 : la fonction *array_push($tableau,valeur)* ajoute *valeur* comme dernier élément de $*tableau ;*
- ligne 49 : la fonction *array_unshift($tableau,valeur)* ajoute *valeur* comme premier élément de $*tableau ;*

1.3.6.2 Le dictionnaire ou tableau associatif

Le script **[bases-06.php]** est le suivant :

```
1.  <?php
2.
3.  // dictionnaires
4.  $conjoints = ["Pierre" => "Gisèle", "Paul" => "Virginie", "Jacques" => "Lucette", "Jean" => ""];
5.  // parcours - 1
6.  print "Nombre d'éléments du dictionnaire : " . count($conjoints) . "\n";
7.  reset($conjoints);
8.  while (list($clé, $valeur) = each($conjoints)) {
9.    print "conjoints[$clé]=$valeur\n";
10. }
11. // tri du dictionnaire sur la clé
12. ksort($conjoints);
13. // parcours - 2
14. reset($conjoints);
15. while (list($clé, $valeur) = each($conjoints)) {
16.   print "conjoints[$clé]=$valeur\n";
17. }
18. // liste des clés du dictionaire
19. $clés = array_keys($conjoints);
20. for ($i = 0; $i < count($clés); $i++) {
21.   print "clés[$i]=$clés[$i]\n";
22. }
23. // liste des valeurs du dictionaire
24. $valeurs = array_values($conjoints);
25. for ($i = 0; $i < count($valeurs); $i++) {
26.   print "valeurs[$i]=$valeurs[$i]\n";
27. }
28. // recherche d'une clé
29. existe($conjoints, "Jacques");
30. existe($conjoints, "Lucette");
31. existe($conjoints, "Jean");
32. // suppression d'une clé-valeur
33. unset($conjoints["Jean"]);
34. print "Nombre d'éléments du dictionnaire : " . count($conjoints) . "\n";
35. foreach ($conjoints as $clé => $valeur) {
36.   print "conjoints[$clé]=$valeur\n";
37. }
38. // fin
39. exit;
40.
41. function existe($conjoints, $mari) {
42.   // vérifie si la clé $mari existe dans le dictionnaire $conjoints
43.   if (isset($conjoints[$mari])) {
44.     print "La clé [$mari] existe associée à la valeur [$conjoints[$mari]]\n";
45.   } else {
46.     print "La clé [$mari] n'existe pas\n";
47.   }
48. }
1
```

Les résultats :

```
1   Nombre d'éléments du dictionnaire : 4
2
3   Deprecated: The each() function is deprecated. This message will be suppressed on further calls in
    C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_05.php on line 8
4   conjoints[Pierre]=Gisèle
5   conjoints[Paul]=Virginie
6   conjoints[Jacques]=Lucette
7   conjoints[Jean]=
8   conjoints[Jacques]=Lucette
9   conjoints[Jean]=
10  conjoints[Paul]=Virginie
11  conjoints[Pierre]=Gisèle
12  clés[0]=Jacques
13  clés[1]=Jean
14  clés[2]=Paul
15  clés[3]=Pierre
16  valeurs[0]=Lucette
17  valeurs[1]=
18  valeurs[2]=Virginie
19  valeurs[3]=Gisèle
```

```
20  La clé [Jacques] existe associée à la valeur [Lucette]
21  La clé [Lucette] n'existe pas
22  La clé [Jean] existe associée à la valeur []
23  Nombre d'éléments du dictionnaire : 3
24  conjoints[Jacques]=Lucette
25  conjoints[Paul]=Virginie
26  conjoints[Pierre]=Gisèle
```

Commentaires

Le code précédent applique à un dictionnaire ce qui a été vu auparavant pour un simple tableau. Nous ne commentons que les nouveautés :

- ligne 12 : la fonction *ksort (key sort)* permet de trier un dictionnaire dans l'ordre naturel de la clé ;
- ligne 19 : la fonction *array_keys($dictionnaire)* rend la liste des clés du dictionnaire sous forme de tableau ;
- ligne 24 : la fonction *array_values($dictionnaire)* rend la liste des valeurs du dictionnaire sous forme de tableau ;
- ligne 43 : la fonction *isset($variable)* rend TRUE si la variable $*variable* a été définie, FALSE sinon ;
- ligne 33 : la fonction *unset($variable)* supprime la variable $*variable*.

1.3.6.3 Les tableaux à plusieurs dimensions

Le script **[bases-07.php]** est le suivant :

```php
<?php

// tableaux classiques multidimensionnels
// initialisation
$multi = array(array(0, 1, 2), array(10, 11, 12, 13), array(20, 21, 22, 23, 24));
// parcours
for ($i1 = 0; $i1 < count($multi); $i1++) {
  for ($i2 = 0; $i2 < count($multi[$i1]); $i2++) {
    print "multi[$i1][$i2]=" . $multi[$i1][$i2] . "\n";
  }
}
// dictionnaires multidimensionnels
// initialisation
$multi = array("zéro" => array(0, 1, 2), "un" => array(10, 11, 12, 13), "deux" => array(20, 21, 22, 23, 24)
);
// parcours
foreach ($multi as $clé => $valeur) {
  for ($i2 = 0; $i2 < count($valeur); $i2++) {
    print "multi[$clé][$i2]=" . $multi[$clé][$i2] . "\n";
  }
}
```

Résultats :

```
1   multi[0][0]=0
2   multi[0][1]=1
3   multi[0][2]=2
4   multi[1][0]=10
5   multi[1][1]=11
6   multi[1][2]=12
7   multi[1][3]=13
8   multi[2][0]=20
9   multi[2][1]=21
10  multi[2][2]=22
11  multi[2][3]=23
12  multi[2][4]=24
13  multi[zéro][0]=0
14  multi[zéro][1]=1
15  multi[zéro][2]=2
16  multi[un][0]=10
17  multi[un][1]=11
18  multi[un][2]=12
19  multi[un][3]=13
20  multi[deux][0]=20
21  multi[deux][1]=21
22  multi[deux][2]=22
23  multi[deux][3]=23
24  multi[deux][4]=24
```

Commentaires

- ligne 5 : les éléments du tableau $*multi* sont eux-mêmes des tableaux ;
- ligne 14 : le tableau $*multi* devient un dictionnaire (*clé,valeur*) où chaque *valeur* est un tableau ;

1.3.7 Les chaînes de caractères

1.3.7.1 Notation

Le script **[bases-08.php]** est le suivant :

```
1.  <?php
2.
3.  // notation des chaînes
4.  $chaine1 = "un";
5.  $chaine2 = 'un';
6.  print "[$chaine1,$chaine2]\n";
```

Résultats :

```
1    [un,un]
```

1.3.7.2 Comparaison

Le script **[bases-09.php]** est le suivant :

```
1.  <?php
2.
3.  // respect strict du type des paramètres des fonctions
4.  declare(strict_types=1);
5.
6.  // fonction de comparaison
7.  function compareModele2Chaine(string $chaine1, string $chaine2): void {
8.    // compare chaine1 et chaine2
9.    if ($chaine1 === $chaine2) {
10.     print "[$chaine1] est égal à [$chaine2]\n";
11.   } else {
12.     print "[$chaine1] est différent de [$chaine2]\n";
13.   }
14. }
15.
16. // tests de comparaisons de chaînes
17. compareModele2Chaine("abcd", "abcd");
18. compareModele2Chaine("", "");
19. compareModele2Chaine("1", "");
20. exit;
```

Résultats :

```
1   [abcd] est égal à [abcd]
2   [] est égal à []
3   [1] est différent de []
```

Commentaires

- ligne 9 du code : on aurait pu utiliser le comparateur == plutôt que ===. Ce dernier opérateur est plus contraignant en ce sens qu'il impose que les deux opérandes soient de même type. A noter qu'ici, il pouvait être remplacé par l'opérateur == puisque le type des deux paramètres est fixé à **string** dans la signature de la fonction ;

1.3.7.3 Liens entre chaînes et tableaux

Le script **[bases-10.php]** est le suivant :

```
1.  <?php
2.
3.  // chaîne vers tableau
4.  $chaine = "1:2:3:4";
5.  $tab = explode(":", $chaine);
6.  // parcours tableau
7.  print "tab a " . count($tab) . " éléments\n";
8.  for ($i = 0; $i < count($tab); $i++) {
9.    print "tab[$i]=$tab[$i]\n";
```

```
10. }
11. // tableau vers chaîne
12. $chaine2 = implode(":", $tab);
13. print "chaine2=$chaine2\n";
14. // ajoutons un champ vide
15. $chaine .= ":";
16. print "chaîne=$chaine\n";
17. $tab = explode(":", $chaine);
18. // parcours tableau
19. print "tab a " . count($tab) . " éléments\n";
20. for ($i = 0; $i < count($tab); $i++) {
21.   print "tab[$i]=$tab[$i]\n";
22. } // on a maintenant 5 éléments, le dernier étant vide
23. // ajoutons de nouveau un champ vide
24. $chaine .= ":";
25. print "chaîne=$chaine\n";
26. $tab = explode(":", $chaine);
27. // parcours tableau
28. print "tab a " . count($tab) . " éléments\n";
29. for ($i = 0; $i < count($tab); $i++) {
30.   print "tab[$i]=$tab[$i]\n";
31. } // on a maintenant 6 éléments, les deux derniers étant vides
32.
```

Résultats :

```
1   tab a 4 éléments
2   tab[0]=1
3   tab[1]=2
4   tab[2]=3
5   tab[3]=4
6   chaine2=1:2:3:4
7   chaîne=1:2:3:4:
8   tab a 5 éléments
9   tab[0]=1
10  tab[1]=2
11  tab[2]=3
12  tab[3]=4
13  tab[4]=
14  chaîne=1:2:3:4::
15  tab a 6 éléments
16  tab[0]=1
17  tab[1]=2
18  tab[2]=3
19  tab[3]=4
20  tab[4]=
21  tab[5]=
```

Commentaires

- ligne 5 : la fonction *explode($séparateur,$chaine)* permet de récupérer les champs de $*chaine* séparés par $*séparateur*. Ainsi *explode(":",$chaine)* permet de récupérer sous forme de tableau les éléments de $*chaine* qui sont séparés par la chaîne ":" ;
- ligne 12 : la fonction *implode*($séparateur,$tableau) fait l'opération inverse de la fonction *explode*. Elle rend une chaîne de caractères formée des éléments de $*tableau* séparés par $*séparateur ;*

1.3.7.4 Les expressions régulières

Le script **[bases-11.php]** est le suivant :

```
1.  <?php
2.
3.  // type strict pour les paramètres de fonctions
4.  declare (strict_types=1);
5.
6.  // expressions régulières en php
7.  // récupérer les différents champs d'une chaîne
8.  // le modèle : une suite de chiffres entourée de caractères quelconques
9.  // on ne veut récupérer que la suite de chiffres
10. $modèle = "/(\d+)/";
11. // on confronte la chaîne au modèle
12. compareModele2Chaine($modèle, "xyz1234abcd");
13. compareModele2Chaine($modèle, "12 34");
14. compareModele2Chaine($modèle, "abcd");
15.
16. // le modèle : une suite de chiffres entourée de caractères quelconques
```

```
17. // on veut la suite de chiffres ainsi que les champs qui suivent et précèdent
18. $modèle = "/^(.*?)(\d+)(.*?)$/";
19. // on confronte la chaîne au modèle
20. compareModele2Chaine($modèle, "xyz1234abcd");
21. compareModele2Chaine($modèle, "12 34");
22. compareModele2Chaine($modèle, "abcd");
23.
24. // le modèle - une date au format jj/mm/aa
25. $modèle = "/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/";
26. compareModele2Chaine($modèle, "10/05/97");
27. compareModele2Chaine($modèle, "  04/04/01  ");
28. compareModele2Chaine($modèle, "5/1/01");
29.
30. // le modèle - un nombre décimal
31. $modèle = "/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/";
32. compareModele2Chaine($modèle, "187.8");
33. compareModele2Chaine($modèle, "-0.6");
34. compareModele2Chaine($modèle, "4");
35. compareModele2Chaine($modèle, ".6");
36. compareModele2Chaine($modèle, "4.");
37. compareModele2Chaine($modèle, " + 4");
38.
39. // fin
40. exit;
41.
42. // ------------------------------------------------------------------------
43. function compareModele2Chaine(string $modèle, string $chaîne): void {
44.   // compare la chaîne $chaîne au modèle $modèle
45.   // on confronte la chaîne au modèle
46.   $champs = [];
47.   $correspond = preg_match($modèle, $chaîne, $champs);
48.   // affichage résultats
49.   print "\nRésultats($modèle,$chaîne)\n";
50.   if ($correspond) {
51.     for ($i = 0; $i < count($champs); $i++) {
52.       print "champs[$i]=$champs[$i]\n";
53.     }
54.   } else {
55.     print "La chaîne [$chaîne] ne correspond pas au modèle [$modèle]\n";
56.   }
57. }
22
```

Résultats :

```
1   Résultats(/(\d+)/,xyz1234abcd)
2   champs[0]=1234
3   champs[1]=1234
4
5   Résultats(/(\d+)/,12 34)
6   champs[0]=12
7   champs[1]=12
8
9   Résultats(/(\d+)/,abcd)
10  La chaîne [abcd] ne correspond pas au modèle [/(\d+)/]
11
12  Résultats(/^(.*?)(\d+)(.*?)$/,xyz1234abcd)
13  champs[0]=xyz1234abcd
14  champs[1]=xyz
15  champs[2]=1234
16  champs[3]=abcd
17
18  Résultats(/^(.*?)(\d+)(.*?)$/,12 34)
19  champs[0]=12 34
20  champs[1]=
21  champs[2]=12
22  champs[3]= 34
23
24  Résultats(/^(.*?)(\d+)(.*?)$/,abcd)
25  La chaîne [abcd] ne correspond pas au modèle [/^(.*?)(\d+)(.*?)$/]
26
27  Résultats(/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/,10/05/97)
28  champs[0]=10/05/97
29  champs[1]=10
30  champs[2]=05
31  champs[3]=97
32
```

```
33  Résultats(/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/,  04/04/01  )
34  champs[0]=  04/04/01
35  champs[1]=04
36  champs[2]=04
37  champs[3]=01
38
39  Résultats(/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/,5/1/01)
40  La chaîne [5/1/01] ne correspond pas au modèle [/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/]
41
42  Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,187.8)
43  champs[0]=187.8
44  champs[1]=
45  champs[2]=187.8
46
47  Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,-0.6)
48  champs[0]=-0.6
49  champs[1]=-
50  champs[2]=0.6
51
52  Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,4)
53  champs[0]=4
54  champs[1]=
55  champs[2]=4
56
57  Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,.6)
58  champs[0]=.6
59  champs[1]=
60  champs[2]=.6
61
62  Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,4.)
63  champs[0]=4.
64  champs[1]=
65  champs[2]=4.
66
67  Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/, + 4)
68  champs[0]= + 4
69  champs[1]=+
70  champs[2]=4
```

Commentaires

- nous utilisons ici les expression régulières pour récupérer les divers champs d'une chaîne de caractères. Les expressions régulières permettent de dépasser les limites de la fonction *implode*. Le principe est de comparer une chaîne de caractères à une autre chaîne appelée *modèle* à l'aide de la fonction *preg_match* :

```
$correspond = preg_match($modèle, $chaîne, $champs);
```

La fonction *preg_match* rend un booléen TRUE si le *modèle* peut être trouvé dans la *chaîne*. Si oui, $*champs*`[0]` représente la sous-chaîne correspondant au modèle. Par ailleurs, si *modèle* contient des sous-modèles entre parenthèses, $*champs*`[1]` est le morceau de $*chaîne* correspondant au 1er sous-modèle, $*champs*`[2]` est le morceau de $*chaîne* correspondant au 2e sous-modèle, etc…

Considérons le 1er exemple. Le modèle est défini ligne 10 : il désigne une suite de un ou plusieurs (+) chiffres (\d) placés n'importe où dans une chaîne. Par ailleurs, le modèle définit un sous-modèle entouré de parenthèses ;

- ligne 12 : le modèle **/(\d+)/** (suite d'un ou plusieurs chiffres n'importe où dans la chaîne) est comparé à la chaîne "**xyz1234abcd**". On voit que la sous-chaîne 1234 correspond au modèle. On aura donc $champs`[0]` égal à "1234". Par ailleurs, le modèle a des sous-modèles entre parenthèses. On aura $champs`[1]`="1234" ;
- ligne 13 : le modèle **/(\d+)/** est comparé à la chaîne "**12 34**". On voit que les sous-chaînes 12 et 34 correspondent au modèle. La comparaison s'arrête à la première sous-chaîne correspondant au modèle. On aura donc, $champs`[0]`=12 et $champs`[1]`=12 ;
- ligne 14 : le modèle **/(\d+)/** est comparé à la chaîne "**abcd**". Aucune correspondance n'est trouvée ;

Explicitons les modèles utilisés dans la suite du code :

```
$modèle = "/^(.*?)(\d+)(.*?)$/";
```

correspond à début de chaîne (^), puis 0 ou plusieurs (*) caractères quelconques (.) puis 1 ou plusieurs (+) chiffres, puis de nouveau 0 ou plusieurs (*) caractères quelconques (.). Le modèle (.*) désigne 0 ou plusieurs caractères quelconques. Un tel modèle va correspondre à n'importe quelle chaîne. Ainsi le modèle /^(.*)(\d+)(.*)$/ ne sera-t-il jamais trouvé car le premier sous-modèle (.*) va absorber toute la chaîne. Le modèle (.*?)(\d+) désigne lui 0 ou plusieurs caractères quelconques **jusqu'au sous-modèle suivant (?)**, ici \d+. Donc les chiffres ne sont maintenant plus absorbés par le

modèle (.*). Le modèle ci-dessus correspond donc à **[début de chaîne (^), une suite de caractères quelconques (.*?), une suite d'un ou plusieurs chiffres (\d+), une suite de caractères quelconques (.*?), la fin de la chaîne ($)]**.

```
$modèle = "/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/";
```

correspond à **[début de chaîne (^), 2 chiffres (\d\d), le caractère / (\/), 2 chiffres, /, 2 chiffres, une suite de 0 ou plusieurs espaces (\s*), la fin de chaîne ($)]**.

```
$modèle = "/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/";
```

correspond à début de chaîne (^), 0 ou plusieurs espaces (\s*), un signe + ou - **[+|-]** présent 0 ou 1 fois (?), une suite de 0 ou plusieurs espaces (\s*), 1 ou plusieurs chiffres suivis d'un point décimal suivi de zéro ou plusieurs chiffres (\d+\.\d*) ou (|) un point décimal (\.) suivi d'un ou plusieurs chiffres (\d+) ou (|) un ou plusieurs chiffres (\d+), une suite de 0 ou plusieurs espaces (\s*)].

Note : le terme **[espace]** dans les expressions régulières désigne un ensemble de caractères : blanc, saut de ligne \n, tabulation \t, retour à la ligne \r, saut de page \f…

1.3.8 Les fonctions

1.3.8.1 Mode de passage des paramètres

Le script **[base-12.php]** est le suivant :

```
1.  <?php
2.
3.  // mode de pasage des paramètres d'une fonction
4.  // respect strict du type des paramères
5.  declare(strict_types=1);
6.
7.  function f(int &$i, int $j): void {
8.    // $i sera obtenu par référence
9.    // $j sera obtenu par valeur
10.   $i++;
11.   $j++;
12.   print "f[i,j]=[$i,$j]\n";
13. }
14.
15. // tests
16. $i = 0;
17. $j = 0;
18. // $i et $j sont passés à la fonction f
19. f($i, $j);
20. print "test[i,j]=[$i,$j]\n";
```

Résultats :

```
1   f[i,j]=[1,1]
2   test[i,j]=[1,0]
```

Commentaires

Le code ci-dessus montre les deux modes de passage de paramètres à une fonction. Prenons l'exemple suivant :

```
1   function f(&$a,$b){
2   …
3   }
4
5   // programme principal
6   $i=10; $j=20;
7   f($i,$j);
```

- ligne 1 : définit les paramètres formels $a et $b de la fonction f. Celle-ci manipule ces deux paramètres formels et rend un résultat ;
- ligne 7 : appel de la fonction f avec deux paramètres effectifs $i et $j. Les liens entre les paramètres formels ($a,$b) et les paramètres effectifs ($i,$j) sont définis par les lignes 1 et 7 :
    - &$a : le signe **&** indique que le paramètre formel $a prendra pour valeur l'adresse du paramètre effectif $i. Dit autrement, $a et $i sont deux références sur un même emplacement mémoire. Manipuler le paramètre formel $a revient à manipuler le paramètre effectif $i. C'est ce que montre l'exécution du code. Ce mode de passage convient

aux paramètres de sortie et aux données volumineuses telles que les tableaux et dictionnaires. On appelle ce mode passage, passage par **référence**.

- $b : le paramètre formel $b prendra pour valeur celle du paramètre effectif $j. C'est un passage par **valeur**. Les paramètres formels et effectifs sont deux variables différentes. Manipuler le paramètre formel $b n'a aucune incidence sur le paramètre effectif $j. C'est ce que montre l'exécution du code. Ce mode de passage convient aux paramètres d'entrée.
- Soit la fonction *échange* qui admet deux paramètres formels $*a* et $*b*. La fonction échange la valeur de ces deux paramètres. Ainsi lors d'un appel échange *($i,$j)*, le code appelant s'attend à ce que les valeurs des deux paramètres effectifs soient échangées. Ce sont donc des paramètres de sortie (ils sont modifiés). On écrira donc :

```
function échange(&$a,&$b){….}
```

Le script suivant **[base-13.php]** montre d'autres exemples :

```
1.  <?php
2.
3.  // types en mode strict
4.  declare(strict_types=1);
5.
6.  // mode de pasage des paramètres
7.
8.  function f(&$i, $j) {
9.    // $i sera obtenu par référence
10.   // $j sera obtenu par valeur
11.   $i++;
12.   $j++;
13.   print "f[i,j]=[$i,$j]\n";
14. }
15.
16. function g(int &$i, int $j): void {
17.   // $i sera obtenu par référence
18.   // $j sera obtenu par valeur
19.   $i++;
20.   $j++;
21.   print "g[i,j]=[$i,$j]\n";
22. }
23.
24. // tests
25. $i = 0;
26. $j = 0;
27. // $i et $j sont passés à la fonction f
28. f($i, $j);
29. print "test[i,j]=[$i,$j]\n";
30. // $i et $j sont passés à la fonction g
31. g($i, $j);
32. print "test[i,j]=[$i,$j]\n";
33. // on passe des paramètres incorrects à f
34. $a = 5.3;
35. $b = 6.2;
36. f($a, $b);
37. print "test[a,b]=[$a,$b]\n";
38. // on passe des paramètres incorrects à f
39. $a = 5.3;
40. $b = 6.2;
41. g($a, $b);
42. print "test[a,b]=[$a,$b]\n";
```

Commentaires

- lignes 8-14 : la fonction **f** étudiée dans le paragraphe précédent mais on n'a pas typé les paramètres ;
- lignes 16-22 : la fonction **g** fait la même chose que la fonction **f** mais on précise le type des paramètres attendus – c'est une nouveauté PHP 7. On attend deux paramètres de type **int.** On veut voir ce qui se passe lorsque le paramètre effectif passé à la fonction n'a pas le type attendu par celle-ci ;
- lignes 25-26 : $i et $j sont deux entiers ;
- lignes 28-29 : appel de la fonction **f** avec des paramètres du type attendu ;
- lignes 31-32 : appel de la fonction **g** avec des paramètres du type attendu ;
- lignes 34-35 : les variables $a et $b sont de type **float** ;
- lignes 36-37 : appel de la fonction **f** avec des paramètres qui ne sont pas du type attendu ;
- lignes 41-42 : appel de la fonction **g** avec des paramètres qui ne sont pas du type attendu ;

Résultats

```
1   f[i,j]=[1,1]
2   test[i,j]=[1,0]
3   g[i,j]=[2,1]
4   test[i,j]=[2,0]
5   f[i,j]=[6.3,7.2]
6   test[a,b]=[6.3,6.2]
7   g[i,j]=[6,7]
8   test[a,b]=[6,6.2]
```

- les lignes 5-6 montrent que la fonction **f** a accepté les deux paramètres de type **float** et a travaillé avec ;
- les lignes 7-8 montrent que la fonction **g** a accepté les deux paramètres de type **float** mais qu'elle les a transformés en type **int** (ligne 7) ;

Maintenant décommentons la ligne 4 :

```
declare(strict_types = 1);
```

Cette instruction indique que les types des paramètres formels doivent être respectés. Si ce n'est pas le cas, une erreur est signalée. Les résultats de l'exécution deviennent alors :

```
1   f[i,j]=[1,1]
2   test[i,j]=[1,0]
3   g[i,j]=[2,1]
4   test[i,j]=[2,0]
5   f[i,j]=[6.3,7.2]
6   test[a,b]=[6.3,6.2]
7
8   Fatal error: Uncaught TypeError: Argument 1 passed to g() must be of the type integer, float given, called
    in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_111.php on line 40 and defined in C:\Data\st-
    2019\dev\php7\php5-exemples\exemples\exemple_111.php:15
9   Stack trace:
10  #0 C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_111.php(40): g(5.3, 6.2)
11  #1 {main}
12    thrown in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_111.php on line 15
```

- lignes 9-10 : l'interpréteur PHP 7 a lancé une exception pour indiquer que le 1er paramètre passé à la fonction **g** n'avait pas le bon type. Il est recommandé d'être strict sur les types des paramètres, chaque fois que c'est possible, pour détecter les erreurs d'appel de fonctions ;

1.3.8.2 Résultats rendus par une fonction

Le script **[base-15.php]** est le suivant :

```
1.  <?php
2.
3.  // types en mode strict
4.  declare(strict_types=1);
5.
6.  // résultats rendus par une fonction
7.  // une fonction peut rendre plusieurs valeurs dans un tableau
8.  list($res1, $res2, $res3) = f1(10);
9.  print "[$res1,$res2,$res3]\n";
10. $res = f1(10);
11. for ($i = 0; $i < count($res); $i++) {
12.   print "f1 : res[$i]=$res[$i]\n";
13. }
14.
15. // une fonction peut rendre un objet
16. $res = f2(10);
17. print "f2 : [$res->res1,$res->res2,$res->res3]\n";
18. // objet de quelle nature ?
19. print "nature de l'objet : ";
20. var_dump($res);
21. print "\n";
22.
23. // on fait la même chose avec la fonction f3
24. $res = f3(10);
25. print "f3 : [$res->res1,$res->res2,$res->res3]\n";
26. // objet de quelle nature ?
27. print "nature de l'objet : ";
28. var_dump($res);
29. print "\n";
30.
```

```
31. // fin
32. exit;
33.
34. // fonction f1
35. function f1(int $valeur): array {
36.   // rend un tableau ($valeur+1,$valeur+2,$valeur+3)
37.   return array($valeur + 1, $valeur + 2, $valeur + 3);
38. }
39.
40. // fonction f2
41. function f2(int $valeur): object {
42.   // rend un objet ($valeur+1,$valeur+2,$valeur+3)
43.   $res->res1 = $valeur + 1;
44.   $res->res2 = $valeur + 2;
45.   $res->res3 = $valeur + 3;
46.   // rend l'objet
47.   return $res;
48. }
49.
50. // fonction f3 - fait la même chose que la fonction f2
51. function f3(int $valeur): object {
52.   // rend un objet ($valeur+1,$valeur+2,$valeur+3)
53.   $res = new stdclass();
54.   $res->res1 = $valeur + 1;
55.   $res->res2 = $valeur + 2;
56.   $res->res3 = $valeur + 3;
57.   // rend l'objet
58.   return $res;
59. }
```

Résultats

```
1   [11,12,13]
2   f1 : res[0]=11
3   f1 : res[1]=12
4   f1 : res[2]=13
5
6   Warning: Creating default object from empty value in C:\Data\st-2019\dev\php7\php5-
    exemples\exemples\bases\base-15.php on line 43
7   f2 : [11,12,13]
8   nature de l'objet : object(stdClass)#1 (3) {
9     ["res1"]=>
10    int(11)
11    ["res2"]=>
12    int(12)
13    ["res3"]=>
14    int(13)
15  }
16
17  f3 : [11,12,13]
18  nature de l'objet : object(stdClass)#2 (3) {
19    ["res1"]=>
20    int(11)
21    ["res2"]=>
22    int(12)
23    ["res3"]=>
24    int(13)
25  }
```

Commentaires

- le programme précédent montre qu'une fonction PHP peut rendre un ensemble de résultats et non un seul, sous la forme d'un tableau ou d'un objet. La notion d'objet est explicitée un peu plus loin ;
- lignes 35-38 : la fonction f1 rend plusieurs valeurs sous la forme d'un tableau (**array**) ;
- lignes 41-48 : la fonction f2 rend plusieurs valeurs sous la forme d'un objet (**object**) ;
- lignes 51-59 : la fonction f3 est identique à la fonction f2 si ce n'est qu'elle crée explicitement un objet, ligne 53 ;
- la ligne 6 des résultats émet un avertissement (warning) indiquant que PHP a été obligé de créer un objet par défaut à la ligne 43 du code, ç-a-d lors de l'utilisation de la notation **[$res→res1]**. La fonction **var_dump** de la ligne 20 du code donne accès à la nature de l'objet et à son contenu. Dans les résultats, on voit que :
  - ligne 8 : l'objet créé par défaut est de type **stdClass** ;
  - lignes 9-10 : la propriété *res1* est de type *entier* et a la valeur 11 ;
  - etc…
  - pour éviter le warning de la ligne 6 des résultats, on crée explicitement, ligne 53 de la fonction f3, un objet de type **stdClass ;**

1.3.9 Les fichiers texte

Le script **[bases-16.php]** est le suivant :

```
1.  <?php
2.
3.  // respect strict du type des paramères des fonctions
4.  declare (strict_types=1);
5.
6.  // exploitation séquentielle d'un fichier texte
7.  // celui-ci est un ensemble de lignes de la forme login:pwd:uid:gid:infos:dir:shell
8.  // chaque ligne est mis dans un dictionnaire sous la forme login => uid:gid:infos:dir:shell
9.  // on fixe le nom du fichier
10. $INFOS = "infos.txt";
11. // on l'ouvre en création
12. if (!$fic = fopen($INFOS, "w")) {
13.   print "Erreur d'ouverture du fichier $INFOS en écriture\n";
14.   exit;
15. }
16. // on génère un contenu arbitraire
17. for ($i = 0; $i < 100; $i++) {
18.   fputs($fic, "login$i:pwd$i:uid$i:gid$i:infos$i:dir$i:shell$i\n");
19. }
20. // on ferme le fichier
21. fclose($fic);
22.
23. // on l'exploite - fgets garde la marque de fin de ligne
24. // cela permet de ne pas récupérer une chaîne vide lors de la lecture d'une ligne blanche
25. // on l'ouvre en lecture
26. if (!$fic = fopen($INFOS, "r")) {
27.   print "Erreur d'ouverture du fichier $INFOS en lecture\n";
28.   exit;
29. }
30.
31. // les lignes font moins de 1000 caractères
32. // la lecture de la ligne s'arrête sur la marque de fin de ligne
33. // ou celle de fin de fichier
34. while ($ligne = fgets($fic, 1000)) {
35.   // on supprime la marque de fin de ligne si elle existe
36.   $ligne = cutNewLineChar($ligne);
37.   // on met la ligne dans un tableau
38.   $infos = explode(":", $ligne);
39.   // on récupère le login
40.   $login = array_shift($infos);
41.   // on néglige le pwd
42.   array_shift($infos);
43.   // on crée une entrée dans le dictionnaire
44.   $dico[$login] = $infos;
45. }
46. // on le ferme
47. fclose($fic);
48.
49. // exploitation du dictionnaire
50. afficheInfos($dico, "login10");
51. afficheInfos($dico, "X");
52.
53. // fin
54. exit;
55.
56. // ----------------------------------------------------------------------------
57. function afficheInfos(array $dico, string $clé): void {
58.   // affiche la valeur associée à clé dans le dictionnaire $dico si elle existe
59.   if (isset($dico[$clé])) {
60.     // valeur existe - est-ce un tableau ?
61.     $valeur = $dico[$clé];
62.     if (is_array($valeur)) {
63.       print "[$clé," . join(":", $valeur) . "]\n";
64.     } else {
65.       // $valeur n'est pas un tableau
66.       print "[$clé,$valeur]\n";
67.     }
68.   } else {
69.     // $clé n'est pas une clé du dictionnaire $dico
70.     print "la clé [$clé] n'existe pas\n";
71.   }
72. }
```

```
73.
74. // ------------------------------------------------------------------------
75. function cutNewLinechar(string $ligne): string {
76.   // on supprime la marque de fin de ligne de $ligne si elle existe
77.   $L = strlen($ligne);  // longueur ligne
78.   while (substr($ligne, $L - 1, 1) == "\n" or substr($ligne, $L - 1, 1) == "\r") {
79.     $ligne = substr($ligne, 0, $L - 1);
80.     $L--;
81.   }
82.   // fin
83.   return($ligne);
84. }
```

Le fichier **infos.txt** :

```
1   login0:pwd0:uid0:gid0:infos0:dir0:shell0
2   login1:pwd1:uid1:gid1:infos1:dir1:shell1
3   login2:pwd2:uid2:gid2:infos2:dir2:shell2
4   …
5   login98:pwd98:uid98:gid98:infos98:dir98:shell98
6   login99:pwd99:uid99:gid99:infos99:dir99:shell99
```

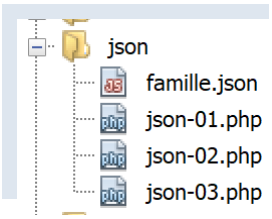
Les résultats :

```
1   [login10,uid10:gid10:infos10:dir10:shell10]
2   la clé [X] n'existe pas
```

Commentaires

- ligne 12 : *fopen(nom_fichier,"w")* ouvre le fichier *nom_fichier* en écriture (*w=write*). Si le fichier n'existe pas, il est créé. S'il existe, il est vidé. Si la création échoue, *fopen* rend la valeur *false*. Dans l'instruction ***if*** *(!$fic = fopen($INFOS, "w"))* {…}, il y a deux opérations successives : 1) *$fic=fopen(..)* 2) *if(! $fic)* {…} ;
- ligne 18 : *fputs($fic,$chaîne)* écrit *chaîne* dans le fichier $*fic*. *$chaine* est écrite avec la marque de fin de ligne \n derrière ;
- ligne 21 : *fclose($fic)* ferme le fichier $*fic* ;
- ligne 26 : *fopen(nom_fichier,"r")* ouvre le fichier *nom_fichier* en lecture (*r=read*). Si l'ouverture échoue (le fichier n'existe pas par exemple), *fopen* rend la valeur *false* ;
- ligne 34 : *fgets($fic,1000)* lit la ligne suivante du fichier dans la limite de 1000 caractères. Dans l'opération ***while*** *($ligne = fgets($fic, 1000))* {…}, il y a deux opérations successives 1) *$ligne=fgets(…)* 2) *while (! $ligne)*. Après que le dernier caractère du fichier a été lu, la fonction *fgets* rend la valeur *false* et la boucle *while* s'arrête. La fonction *fgets* tente ici de lire au plus 1000 caractères mais s'arrête dès qu'une marque de fin de ligne est rencontrée. Ici toutes les lignes ayant moins de 1000 caractères, **[fgets]** lit une ligne de texte, caractère de fin de ligne inclus. La fonction *cutNewLineChar* des lignes 75-84 élimine les éventuels caractères de fin de ligne ;
- ligne 77 : la fonction *strlen($chaîne)* rend le nombre de caractères de $*chaîne* ;
- ligne 78 : la fonction *substr($ligne, $position, $taille)* rend $*taille* caractères de $*ligne*, pris à partir du caractère n° $*position*, le 1er caractère ayant le n° 0. Sur les machines windows, la marque de fin de ligne est "\r\n". Sur les machines Unix, c'est la chaîne "\n" ;
- ligne 40 : la fonction *array_shift($tableau)* élimine le 1er élément de $*tableau* et le rend comme résultat. On néglige ici le résultat rendu par *array_shift* ;
- ligne 62 : la fonction *is_array($variable)* rend *true* si $*variable* est un tableau, *false* sinon ;
- ligne 63 : la fonction *join* fait la même chose que la fonction *implode* déjà rencontrée ;

1.3.10 Encodage / décodage jSON



L'encodage / décodage jSON (**J**avaScript **O**bject **N**otation) est quelque chose que nous allons utiliser intensivement dans l'exercice qui sert de fil rouge au document. Les scripts **[json-01.php, json-02.php, json-03.php]** expliquent ce qui est à savoir pour la suite.

Le script **[json-01.php]** est le suivant :

```
1.  <?php
```

```
2.
3.  $array1 = ["nom" => "séléné", "prénom" => "bénédicte", "âge" => 34];
4.  // encodage json du tableau array1 avec caractères Unicode échappés
5.  print "encodage json du tableau array1 avec caractères Unicode échappés\n";
6.  $json1 = json_encode($array1);
7.  print "json1=$json1\n";
8.  // encodage json du tableau array1 avec caractères Unicode non échappés
9.  print "encodage json du tableau array1 avec caractères Unicode non échappés\n";
10. $json2 = json_encode($array1, JSON_UNESCAPED_UNICODE);
11. print "json2=$json2\n";
12. // décodage JSON dans tableau associatif
13. print "décodage JSON de json2 dans tableau associatif\n";
14. $array2 = json_decode($json2, true);
15. var_dump($array2);
16. foreach ($array2 as $key => $value) {
17.   print "$key:$value\n";
18. }
19. // décodage JSON dans objet
20. print "décodage JSON de json2 dans objet stdClass\n";
21. $array2 = json_decode($json2);
22. var_dump($array2);
23. print "prénom=$array2->prénom\n";
24. print "nom=$array2->nom\n";
25. print "âge=$array2->âge\n";
```

Résultats

```
1   encodage json du tableau array1 avec caractères Unicode échappés
2   json1={"nom":"s\u00e9l\u00e9n\u00e9","pr\u00e9nom":"b\u00e9n\u00e9dicte","\u00e2ge":34}
3   encodage json du tableau array1 avec caractères Unicode non échappés
4   json2={"nom":"séléné","prénom":"bénédicte","âge":34}
5   décodage jSON de json2 dans tableau associatif
6   array(3) {
7     ["nom"]=>
8     string(9) "séléné"
9     ["prénom"]=>
10    string(11) "bénédicte"
11    ["âge"]=>
12    int(34)
13  }
14  nom:séléné
15  prénom:bénédicte
16  âge:34
17  décodage jSON de json2 dans objet stdClass
18  object(stdClass)#1 (3) {
19    ["nom"]=>
20    string(9) "séléné"
21    ["prénom"]=>
22    string(11) "bénédicte"
23    ["âge"]=>
24    int(34)
25  }
26  prénom=bénédicte
27  nom=séléné
28  âge=34
```

Commentaires

- ligne 6 du code : la fonction **[json_encode]** transforme son paramètre en chaîne de caractères jSON ;
- ligne 2 des résultats : la chaîne jSON produite. Les caractères Unicode éâ ont été remplacés par leur code Unicode qui commence par \u ;
- ligne 10 du code : on refait la même chose en demandant cette fois-ci à ce que les caractères Unicode soient conservés tels quels ;
- ligne 4 des résultats : la chaîne jSON résultante. Elle est beaucoup plus lisible ;
- lignes 14-18 du code : on fait l'opération inverse. On transforme une chaîne jSON en tableau associatif ;
- lignes 6-13 des résultats : on voit qu'on a récupéré un tableau associatif ;
- lignes 19-25 du code : on transforme une chaîne jSON en objet de type **[stdClass]** ;
- lignes 18-25 des résultats : on voit qu'on a récupéré un objet de type **[stdClass]** ;
- lignes 23-25 du code : l'attribut A d'un objet O est noté **[O→A]** ;

On peut encoder en jSON des tableaux à plusieurs niveaux comme le montre le script **[json-02.php]** suivant :

```
1.  <?php
```

```
2.
3.  $array = ["nom" => "séléné", "prénom" => "bénédicte", "âge" => 34,
4.    "mari" => ["nom" => "icariù", "prénom" => "ignacio", "âge" => 35],
5.    "enfants" => [
6.      ["prénom" => "angèle", "age" => 8],
7.      ["prénom" => "andré", "age" => 2],
8.    ]];
9.  // encodage JSON du tableau à plusieurs niveaux
10. print "encodage JSON d'un tableau à plusieurs niveaux\n";
11. $json = json_encode($array, JSON_UNESCAPED_UNICODE);
12. print "json=$json\n";
```

Résultats

```
1   encodage jSON d'un tableau à plusieurs niveaux
2   json={"nom":"séléné","prénom":"bénédicte","âge":34,"mari":{"nom":"icariù","prénom":"ignacio","âge":35},"enf
    ants":[{"prénom":"angèle","age":8},{"prénom":"andré","age":2}]}
```

Commentaires

Dans la chaîne jSON :

- les tableaux non associatifs sont entourés de crochets **[]** ;
- les tableaux associatifs sont entourés d'accolades {} ;

Le script **[json-03.php]** montre comment exploiter le fichier jSON **[famille.json]** suivant :

```
1   {
2       "épouse": {
3           "nom": "séléné",
4           "prénom": "bénédicte",
5           "âge": 34
6       },
7       "mari": {
8           "nom": "icariù",
9           "prénom": "ignacio",
10          "âge": 35
11      },
12      "enfants": [
13          {
14              "prénom": "angèle",
15              "age": 8
16          },
17          {
18              "prénom": "andré",
19              "age": 2
20          }
21      ]
22  }
```

Le script **[json-03.php]** est le suivant :

```
1.  <?php
2.
3.  // lecture du fichier JSON
4.  $json = file_get_contents("famille.json");
5.  // décodage json en objet
6.  $famille1 = json_decode($json);
7.  print "----famille1\n";
8.  var_dump($famille1);
9.  // décodage json en tableau associatif
10. print "----famille2\n";
11. $famille2 = json_decode($json, true);
12. var_dump($famille2);
```

Commentaires

- ligne 4 : la fonction **[file_get_contents]** lit le contenu du fichier nommé **[famille.json]** et le met dans la variable **[$json]** ;
- la variable est ensuite décodée en objet (lignes 5-8) et en tableau associatif (lignes 9-12) ;

Résultats

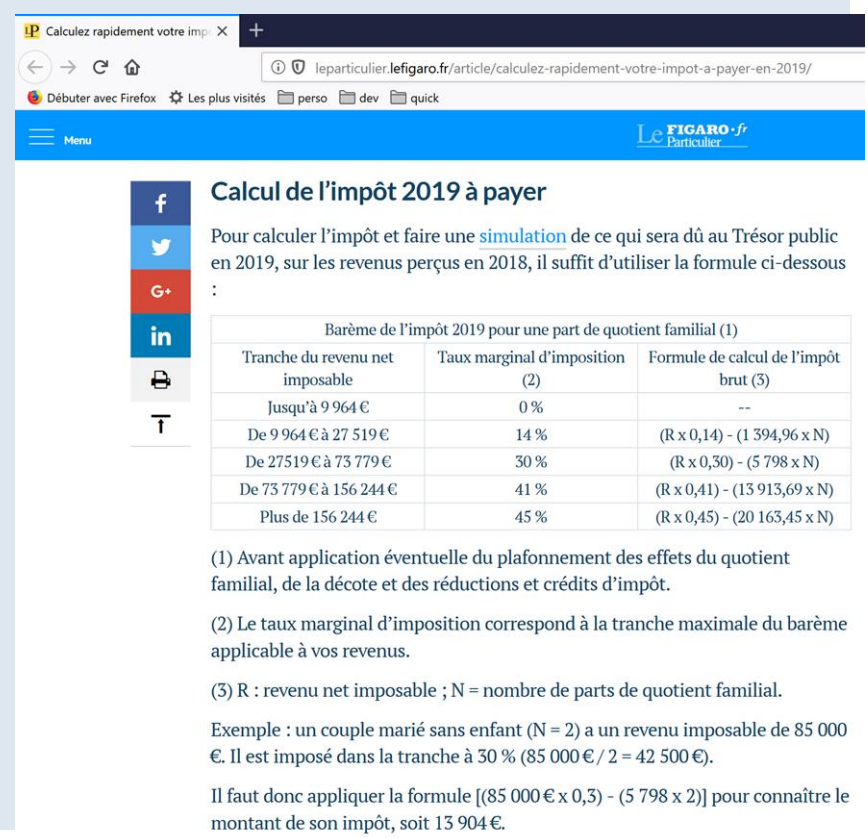
```
----famille1
object(stdClass)#2 (3) {
  ["épouse"]=>
  object(stdClass)#1 (3) {
    ["nom"]=>
    string(9) "séléné"
    ["prénom"]=>
    string(11) "bénédicte"
    ["âge"]=>
    int(34)
  }
  ["mari"]=>
  object(stdClass)#3 (3) {
    ["nom"]=>
    string(7) "icariù"
    ["prénom"]=>
    string(7) "ignacio"
    ["âge"]=>
    int(35)
  }
  ["enfants"]=>
  array(2) {
    [0]=>
    object(stdClass)#4 (2) {
      ["prénom"]=>
      string(7) "angèle"
      ["age"]=>
      int(8)
    }
    [1]=>
    object(stdClass)#5 (2) {
      ["prénom"]=>
      string(6) "andré"
      ["age"]=>
      int(2)
    }
  }
}
----famille2
array(3) {
  ["épouse"]=>
  array(3) {
    ["nom"]=>
    string(9) "séléné"
    ["prénom"]=>
    string(11) "bénédicte"
    ["âge"]=>
    int(34)
  }
  ["mari"]=>
  array(3) {
    ["nom"]=>
    string(7) "icariù"
    ["prénom"]=>
    string(7) "ignacio"
    ["âge"]=>
    int(35)
  }
  ["enfants"]=>
  array(2) {
    [0]=>
    array(2) {
      ["prénom"]=>
      string(7) "angèle"
      ["age"]=>
      int(8)
    }
    [1]=>
    array(2) {
      ["prénom"]=>
      string(6) "andré"
      ["age"]=>
      int(2)
    }
  }
}
```

Commentaires

- lignes 1-38 : l'objet issu du décodage du fichier jSON **[famille.json]** ;
- lignes 39-76 : le tableau associatif issu du décodage du fichier jSON **[famille.json]** ;

1.4 Exercice d'application – versions 1 et 2

1.4.1 Le problème



Calculez rapidement votre imp

leparticulier.lefigaro.fr/article/calculez-rapidement-votre-impot-a-payer-en-2019/

Débuter avec Firefox | Les plus visités | perso | dev | quick

Menu — Le FIGARO.fr Particulier

Calcul de l'impôt 2019 à payer

Pour calculer l'impôt et faire une simulation de ce qui sera dû au Trésor public en 2019, sur les revenus perçus en 2018, il suffit d'utiliser la formule ci-dessous :

Barème de l'impôt 2019 pour une part de quotient familial (1)		
Tranche du revenu net imposable	Taux marginal d'imposition (2)	Formule de calcul de l'impôt brut (3)
Jusqu'à 9 964 €	0 %	--
De 9 964 € à 27 519 €	14 %	(R x 0,14) - (1 394,96 x N)
De 27519 € à 73 779 €	30 %	(R x 0,30) - (5 798 x N)
De 73 779 € à 156 244 €	41 %	(R x 0,41) - (13 913,69 x N)
Plus de 156 244 €	45 %	(R x 0,45) - (20 163,45 x N)

(1) Avant application éventuelle du plafonnement des effets du quotient familial, de la décote et des réductions et crédits d'impôt.

(2) Le taux marginal d'imposition correspond à la tranche maximale du barème applicable à vos revenus.

(3) R : revenu net imposable ; N = nombre de parts de quotient familial.

Exemple : un couple marié sans enfant (N = 2) a un revenu imposable de 85 000 €. Il est imposé dans la tranche à 30 % (85 000 € / 2 = 42 500 €).

Il faut donc appliquer la formule [(85 000 € x 0,3) - (5 798 x 2)] pour connaître le montant de son impôt, soit 13 904 €.

Le tableau ci-dessus permet de calculer l'impôt dans le cas simplifié d'un contribuable n'ayant que son seul salaire à déclarer. Comme l'indique la note (1), l'impôt ainsi calculé est l'impôt avant trois mécanismes :

- le plafonnement du quotient familial qui intervient pour les hauts revenus ;
- la décôte et la réduction d'impôts qui interviennent pour les faibles revenus ;

Ainsi le calcul de l'impôt comprend les étapes suivantes **[http://impotsurlerevenu.org/comprendre-le-calcul-de-l-impot/1217-calcul-de-l-impot-2019.php]** :

CALCUL DE L'IMPÔT 2019 SUR REVENU 2018

L'impôt est calculé en plusieurs étapes :

1. Détermination du **revenu net imposable** : Revenus - Charges déductibles ;
2. Détermination du **quotient familial (QF)** : Revenu net / Parts fiscales ;
3. Calcul de **l'impôt brut** : Application du QF au barème progressif de l'impôt ;
4. Calcul de **l'impôt net** : Application de la décote, du plafonnement du QF et déduction des crédits et réductions d'impôt.

On se propose d'écrire un programme permettant de calculer l'impôt d'un contribuable dans le cas simplifié d'un contribuable n'ayant que son seul salaire à déclarer :

1.4.1.1 Calcul de l'impôt brut

L'impôt brut peut être calculé de la façon suivante :

On calcule d'abord le nombre de parts du contribuable :

- chaque parent amène 1 part ;
- les deux premiers enfants amènent chacun 1/2 part ;
- les enfants suivants amènent une part chacun :

Le nombre de part est donc :
  - **nbParts=1+nbEnfants*0,5+(nbEnfants-2)*0,5** si le salarié n'est pas marié ;
  - **nbParts=2+nbEnfants*0,5+(nbEnfants-2)*0,5** s'il est marié ;

  où *nbEnfants* est son nombre d'enfants ;
- on calcule le revenu imposable **R=0.9*S** où *S* est le salaire annuel ;
- on calcule le quotient familial **QF=R/nbParts ;**
- on calcule l'impôt brut **I** d'après les données suivantes (2019) :

9964	0	0
27519	0.14	1394.96
73779	0.3	5798
156244	0.41	13913.69
0	0.45	20163.45

Chaque ligne a 3 champs : *champ1*, *champ2*, *champ3*. Pour calculer l'impôt I, on recherche la première ligne où *QF*<=*champ1* et on prend les valeurs de cette ligne. Par exemple, pour un salarié marié avec deux enfants et un salaire annuel S de **50000** euros :

Revenu imposable : **R**=0,9*S=45000
Nombre de parts : **nbParts**=2+2*0,5=3
Quotient familial : **QF**=45000/3=15000

La 1re ligne où QF<=champ1 est la suivante :

```
27519           0.14    1394.96
```

L'impôt *I* est alors égal à **0.14*R – 1394,96*nbParts**=**[0,14*45000-1394,96*3]**=**2115**. L'impôt est arrondi à l'euro inférieur.

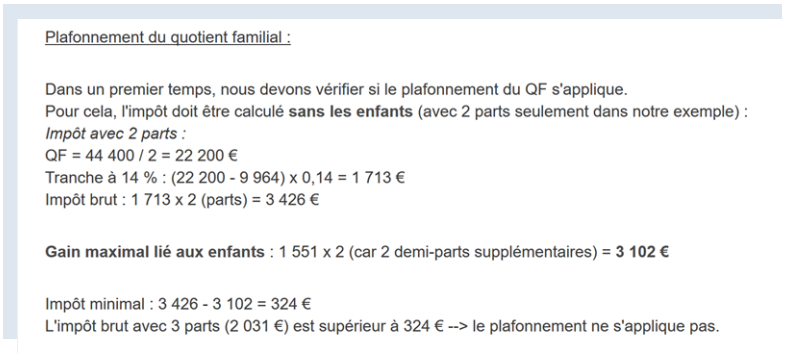
Si la relation *QF*<=*champ1* dès la 1re ligne, alors l'impôt est nul.

Si QF est tel que la relation *QF*<=*champ1* n'est jamais vérifiée, alors ce sont les coefficients de la dernière ligne qui sont utilisés. Ici :

```
0                       0.45            20163.45
```

ce qui donne l'impôt brut I=**0.45*R – 20163,45*nbParts**.

1.4.1.2 Plafonnement du quotient familial



Pour savoir si le plafonnement du quotient familial QF s'applique, on refait le calcul de l'impôt brut sans les enfants. Toujours pour le salarié marié avec deux enfants et un salaire annuel S de **50000** euros :

Revenu imposable : **R**=0,9*S=45000
Nombre de parts : **nbParts**=2 (on ne compte plus les enfants)
Quotient familial : **QF**=45000/2=22500

La 1re ligne où QF<=champ1 est la suivante :

```
    27519           0.14     1394.96
```

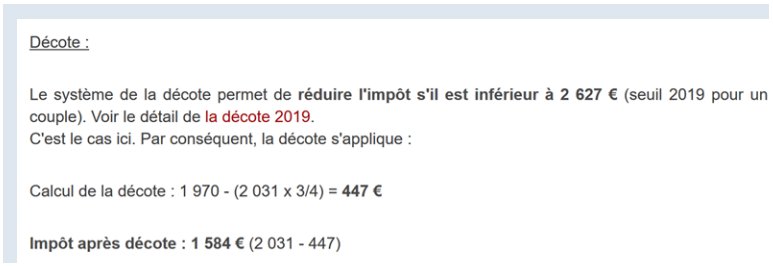
L'impôt *I* est alors égal à **0.14*R – 1394,96*nbParts**=`[0,14*45000-1394,96*2]`=**3510**.

Gain maximal lié aux enfants : 1551 * 2 = 3102 euros
Impôt minimal : 3510-3102 = 408 euros
L'impôt brut avec 3 parts déjà calculé 2115 euros est supérieur à l'impôt minimal 408 euros, donc le plafonnement familial ne s'applique pas ici.

De façon générale, l'impôt brut est **sup(impôt1, impôt2)** où :

- `[impôt1]` : est l'impôt brut calculé avec les enfants ;
- `[impôt2]` : est l'impôt brut calculé sans les enfants et diminué du gain maximal (ici 1551 euros par demi-part) lié aux enfants ;

1.4.1.3 Calcul de la décôte



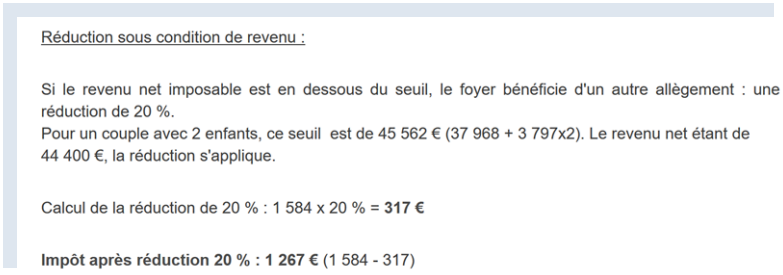
Toujours pour le salarié marié avec deux enfants et un salaire annuel S de **50000** euros :

L'impôt brut (2115) issu de l'étape précédente est inférieur à 2627 euros pour un couple (1595 euros pour un célibataire) : la décôte s'applique donc. Elle est obtenue avec le calcul suivant :

décôte= seuil (couple=1970/célibataire=1196)-0,75* Impôt brut

décôte=1970-0,75*2115=383,75 arrondi à **384** euros.
Nouvel Impôt brut= 2115-384= **1731** euros

1.4.1.4 Calcul de la réduction d'impôts



Au-dessous d'un certain seuil, une réduction de 20 % est faite sur l'impôt brut issu des calculs précédents. En 2019, les seuils sont les suivants :

- célibataire : 21037 euros ;
- couple : 42074 euros ; (le chiffre 37968 utilisé dans l'exemple ci-dessus semble erroné) ;

Ce seuil est augmenté de la valeur : 3797 * (nombre de demi-parts amenées par les enfants).

Toujours pour le salarié marié avec deux enfants et un salaire annuel S de **50000** euros :

- son revenu imposable (45000 euros) est inférieur au seuil (42074+2*3797)=49668 euros ;
- il a donc droit à une réduction réduction de 20 % de son impôt : 1731 * 0,2= 346,2 euros arrondi à 347 euros ;
- l'impôt brut du contribuable devient : 1731-347= **1384** euros ;

1.4.1.5 Calcul de l'impôt net

Notre calcul s'arrêtera là : l'impôt net à payer sera de **1384** euros. Dans la réalité, le contribuable peut bénéficier d'autres réductions notamment pour des dons à des organismes d'intérêt public ou général.

1.4.1.6 Cas des hauts revenus

Notre exemple précédent correspond à la majorité des cas de salariés. Cependant le calcul de l'impôt est différent dans le cas des hauts revenus.

1.4.1.6.1 Plafonnement de la réduction de 10 % sur les revenus annuels

Dans la plupart des cas, le revenu imposable est obtenu par la formule : R=0,9*S où S est le salaire annuel. On appelle cela la réduction des 10 %. Cette réduction est plafonnée. En 2019 :

- elle ne peut être supérieure à 12502 euros ;
- elle ne peut être inférieure à 437 euros ;

Prenons le cas d'un salarié non marié sans enfants et un salaire annuel de 200000 euros :

- la réduction de 10 % est de 20000 euros > 12502 euros. Elle est donc ramenée à 12502 euros ;

1.4.1.6.2 Plafonnement du quotient familial

Prenon un cas où le plafonnement familial présenté au paragraphe lien intervient. Prenons le cas d'un couple avec trois enfants et des revenus annuels de 100000 euros. Reprenons les étapes du calcul :

- l'abattement de 10 % est de 10000 euros < 12502 euros. Le revenu imposable **R** est donc 100000-10000=90000 euros ;
- le couple a **nbParts**=2+0,5*2+1=**4** parts ;
- son quotient familial est donc **QF**= R/nbParts=90000/4=**22500** euros ;
- son impôt brut **I1 avec** enfants est I1=0,14*90000-1394,96*4= 7020 euros ;
- son impôt brut **I2 sans** enfants :
  - **QF**=90000/2=45000 euros ;
  - **I2**=0,3*90000-5798*2=15404 euros ;
  - la règle du plafonnement du quotient familial dit que le gain amené par les enfants ne peut dépasser (1551*4 demi-parts)=6204 euros. Or ici, il est I2-I1=15404-7020= 8384 euros, donc supérieur à 6204 euros ;
  - l'impôt brut est donc recalculé comme **I3**=**I2**-6204=15404-6204= **9200** euros ;

Ce couple n'aura ni décôte, ni réduction et son impôt final sera de **9200** euros.

1.4.1.7 Chiffres officiels

Le calcul de l'impôt est complexe. Tout au long du document, les tests seront faits avec les exemples suivants. Les résultats sont ceux du simulateur de l'administration fiscale **[https://www3.impots.gouv.fr/simulateur/calcul_impot/2019/simplifie/index.htm]** :

Contribuable	Résultats officiels	Résultats de l'algorithme du document
Couple avec 2 enfants et des revenus annuels de 55555 euros	Impôt=**2815** euros Taux d'imposition=14 %	Impôt=**2814** euros Taux d'imposition=14 %
Couple avec 2 enfants et des revenus annuels de 50000 euros	Impôt=**1385** euros Décôte=720 euros Réduction=0 euros Taux d'imposition=14 %	Impôt=**1384** euros Décôte=384 euros Réduction=347 euros Taux d'imposition=14 %
Couple avec 3 enfants et des revenus annuels de 50000 euros	Impôt=**0** euro Décôte=384 euros Réduction=346 euros Taux d'imposition=14 %	Impôt=**0** euro Décôte=720 euros Réduction=0 euro Taux d'imposition=14 %
Célibataire avec 2 enfants et des revenus annuels de 100000 euros	Impôt=**19884** euros Décôte=0 euro Réduction=0 euro Taux d'imposition=41 %	Impôt=**19884** euros Surcôte=4480 euros Décôte=0 euro Réduction=0 euro Taux d'imposition=41 %
Célibataire avec 3 enfants et des revenus annuels de 100000 euros	Impôt=**16782** euros Décôte=0 euro	Impôt=**16782** euros Surcôte=7176 euros

	Réduction=0 euro Taux d'imposition=41 %	Décôte=0 euro Réduction=0 euro Taux d'imposition=41 %
Couple avec 3 enfants et des revenus annuels de 100000 euros	Impôt=**9200** euros Décôte=0 euro Réduction=0 euro Taux d'imposition=30 %	Impôt=**9200** euros Surcôte=2180 euros Décôte=0 euro Réduction=0 euro Taux d'imposition=30 %
Couple avec 5 enfants et des revenus annuels de 100000 euros	Impôt=**4230** euros Décôte=0 euro Réduction=0 euro Taux d'imposition=14 %	Impôt=**4230** euros Décôte=0 euro Réduction=0 euro Taux d'imposition=14 %
Célibataire sans enfants et des revenus annuels de 100000 euros	Impôt=**22986** euros Décôte=0 euro Réduction=0 euro Taux d'imposition=41 %	Impôt= **22986** euros Surcôte=0 euro Décôte=0 euro Réduction=0 euro Taux d'imposition=41 %
Couple avec 2 enfants et des revenus annuels de 30000 euros	Impôt=**0** euro Décôte=0 euro Réduction=0 euro Taux d'imposition=0 %	Impôt=**0** euro Décôte=0 euro Réduction=0 euro Taux d'imposition=0 %
Célibataire sans enfants et des revenus annuels de 200000 euros	Impôt=**64211** euro Décôte=0 euro Réduction=0 euro Taux d'imposition=45 %	Impôt= **64210** euros Surcôte=7498 euros Décôte=0 euro Réduction=0 euro Taux d'imposition=45 %
Couple avec 3 enfants et des revenus annuels de 200000 euros	Impôt=**42843** euro Décôte=0 euro Réduction=0 euro Taux d'imposition=41 %	Impôt=**42842** euros Surcôte=17283 euros Décôte=0 euro Réduction=0 euro Taux d'imposition=41 %

Ci-dessus, on appelle surcôte, ce que paient en plus les hauts revenus à cause de deux phénomènes :

- le plafonnement de l'abattement de 10 % sur les revenus annuels ;
- le plafonnement du quotient familial ;

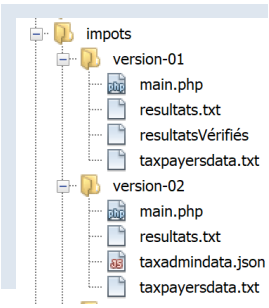
Cet indicateur n'a pu être vérifié car le simulateur de l'administration fiscale ne le donne pas.

On voit que l'algorithme du document donne un impôt juste à chaque fois, avec cependant une marge d'erreur de 1 euro. Cette marge d'erreur provient des arrondis. Toutes les sommes d'argent sont arrondies parfois à l'euro supérieur, parfois à l'euro inférieur. Comme je ne connaissais pas les règles officielles, les sommes d'argent de l'algorithme du document ont été arrondies :

- à l'euro supérieur pour les décôtes et réductions ;
- à l'euro inférieur pour les surcôtes et l'impôt final ;

Dans la suite, des tests seront établis pour vérifier la validité des résultats. Ils seront faits avec les exemples du tableau précédent avec une marge d'erreur acceptée de 1 euro.

1.4.2 L'arborescence des scripts



1.4.3 Version 1

1.4.3.1 L'algorithme

Nous présentons un premier programme où :

- les données nécessaires au calcul de l'impôt sont codées en dur dans le code sous forme de tableaux et de constantes ;
- les données des contribuables (marié, enfants, salaire) sont dans un premier fichier texte **[taxpayersdata.txt]** ;
- les résultats du calcul de l'impôt (marié, enfants, salaire, impôt) sont mémorisés dans un second fichier texte **[resultats.txt]** ;

Le script **[version-01/main.php]** est le suivant :

```
1.  <?php
2.
3.  // types stricts pour les paramètres de fonctions
4.  declare(strict_types=1);
5.
6.  // constantes globales
7.  define("PLAFOND_QF_DEMI_PART", 1551);
8.  define("PLAFOND_REVENUS_CELIBATAIRE_POUR_REDUCTION", 21037);
9.  define("PLAFOND_REVENUS_COUPLE_POUR_REDUCTION", 42074);
10. define("VALEUR_REDUC_DEMI_PART", 3797);
11. define("PLAFOND_DECOTE_CELIBATAIRE", 1196);
12. define("PLAFOND_DECOTE_COUPLE", 1970);
13. define("PLAFOND_IMPOT_COUPLE_POUR_DECOTE", 2627);
14. define("PLAFOND_IMPOT_CELIBATAIRE_POUR_DECOTE", 1595);
15. define("ABATTEMENT_DIXPOURCENT_MAX", 12502);
16. define("ABATTEMENT_DIXPOURCENT_MIN", 437);
17.
18. // définition des constantes locales
19. $DATA = "taxpayersdata.txt";
20. $RESULTATS = "resultats.txt";
21. $limites = array(9964, 27519, 73779, 156244, 0);
22. $coeffR = array(0, 0.14, 0.3, 0.41, 0.45);
23. $coeffN = array(0, 1394.96, 5798, 13913.69, 20163.45);
24.
25. // lecture des données
26. $data = fopen($DATA, "r");
27. if (!$data) {
28.   print "Impossible d'ouvrir en lecture le fichier des données [$DATA]\n";
29.   exit;
30. }
31.
32. // ouverture fichier des résultats
33. $résultats = fopen($RESULTATS, "w");
34. if (!$résultats) {
35.   print "Impossible de créer le fichier des résultats [$RESULTATS]\n";
36.   exit;
37. }
38.
39. // on exploite la ligne courante du fichier des données
40. while ($ligne = fgets($data, 100)) {
41.   // on enlève l'éventuelle marque de fin de ligne
42.   $ligne = cutNewLineChar($ligne);
43.   // on récupère les 3 champs marié:enfants:salaire qui forment $ligne
44.   list($marié, $enfants, $salaire) = explode(",", $ligne);
45.   // on calcule l'impôt
46.   $result = calculImpot($marié, (int) $enfants, (float) $salaire, $limites, $coeffR, $coeffN);
47.   // on inscrit le résultat dans le fichier des résultats
48.   $result = ["marié" => $marié, "enfants" => $enfants, "salaire" => $salaire] + $result;
49.   fputs($résultats, \json_encode($result, JSON_UNESCAPED_UNICODE) . "\n");
50.   // donnée suivante
51. }
52. // on ferme les fichiers
53. fclose($data);
54. fclose($résultats);
55.
56. // fin
57. exit;
58.
59. // ----------------------------------------------------------------------
60. function cutNewLinechar(string $ligne): string {
61.   // on supprime la marque de fin de ligne de $ligne si elle existe
62.   $L = strlen($ligne);  // longueur ligne
63.   while (substr($ligne, $L - 1, 1) === "\n" or substr($ligne, $L - 1, 1) === "\r") {
```

```
64.     $ligne = substr($ligne, 0, $L - 1);
65.     $L--;
66.   }
67.   // fin
68.   return($ligne);
69. }
70.
71. // calcul de l'impôt
72. // ------------------------------------------------------------------------
73. function calculImpot(string $marié, int $enfants, float $salaire, array $limites, array $coeffR, array $coe
    ffN): array {
74.   ..
75.   // résultat
76.   return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" => $réduction,
    "taux" => $taux];
77. }
78.
79. // ------------------------------------------------------------------------
80. function calculImpot2(string $marié, int $enfants, float $salaire, array $limites, array $coeffR, array $co
    effN): array {
81.   …
82.   // résultat
83.   return ["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]];
84. }
85.
86. // revenuImposable=salaireAnnuel-abattement
87. // l'abattement a un min et un max
88. function getRevenuImposable(float $salaire): float {
89.   …
90.   // résultat
91.   return floor($revenuImposable);
92. }
93.
94. // calcule une décôte éventuelle
95. function getDecote(string $marié, float $salaire, float $impots): float {
96.   …
97.   // résultat
98.   return ceil($décôte);
99. }
100.
101.// calcule une réduction éventuelle
102.function getRéduction(string $marié, float $salaire, int $enfants, float $impots): float {
103.  ….
104.  // résultat
105.  return ceil($réduction);
106.}
```

Le fichier des données *taxpayersdata.txt* (marié, enfants, salaire) :

```
1   oui,2,55555
2   oui,2,50000
3   oui,3,50000
4   non,2,100000
5   non,3,100000
6   oui,3,100000
7   oui,5,100000
8   non,0,100000
9   oui,2,30000
10  non,0,200000
11  oui,3,200000
```

Les fichier *résultats.txt* (marié, enfants, salaire, impôt, surcôte, décôte, réduction, taux d'imposition) des résultats obtenus :

```
1   {"marié":"oui","enfants":"2","salaire":"55555","impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.1
    4}
2   {"marié":"oui","enfants":"2","salaire":"50000","impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux"
    :0.14}
3   {"marié":"oui","enfants":"3","salaire":"50000","impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14
    }
4   {"marié":"non","enfants":"2","salaire":"100000","impôt":19884,"surcôte":4480,"décôte":0,"réduction":0,"taux
    ":0.41}
5   {"marié":"non","enfants":"3","salaire":"100000","impôt":16782,"surcôte":7176,"décôte":0,"réduction":0,"taux
    ":0.41}
6   {"marié":"oui","enfants":"3","salaire":"100000","impôt":9200,"surcôte":2180,"décôte":0,"réduction":0,"taux"
    :0.3}
```

```
7   {"marié":"oui","enfants":"5","salaire":"100000","impôt":4230,"surcôte":0,"décôte":0,"réduction":0,"taux":0.
    14}
8   {"marié":"non","enfants":"0","salaire":"100000","impôt":22986,"surcôte":0,"décôte":0,"réduction":0,"taux":0
    .41}
9   {"marié":"oui","enfants":"2","salaire":"30000","impôt":0,"surcôte":0,"décôte":0,"réduction":0,"taux":0}
10  {"marié":"non","enfants":"0","salaire":"200000","impôt":64210,"surcôte":7498,"décôte":0,"réduction":0,"taux
    ":0.45}
11  {"marié":"oui","enfants":"3","salaire":"200000","impôt":42842,"surcôte":17283,"décôte":0,"réduction":0,"tau
    x":0.41}
```

Commentaires

- ligne 4 : on force le respect strict du type des paramètres des fonctions ;
- lignes 7-16 : définition de toutes les constantes nécessaire au calcul de l'impôt ;
- ligne 19 : le nom du fichier texte contenant les données des contribuables (marié, enfants, salaire) ;
- ligne 20 : le nom du fichier texte contenant les résultats (marié, enfants, salaire, impôt) du calcul de l'impôt ;
- lignes 21-23 : les trois tableaux des données définissant les différentes tranches d'imposition du calcul de l'impôt ;
- lignes 26-30 : ouverture en lecture **[r]** du fichier des données contribuables. La fonction **[fopen]** rend le booléen FALSE si l'ouverture n'a pu se faire ;
- lignes 33-37 : ouverture en écriture **[w]** du fichier des résultats ;
- lignes 40-51 : boucle de lecture des lignes (marié, enfants, salaire) du fichier des données contribuables ;
- ligne 40 : la fonction **[fgets]** lit 100 caractères et s'arrête à la 1re marque de fin de ligne rencontrée. Ici toutes les lignes font moins de 100 caractères. Si une marque de fin de ligne a été rencontrée, elle est incluse dans la chaîne rendue. Lorsque la fin du fichier est rencontrée, la fonction **[fgets]** rend la valeur FALSE ;
- ligne 42 : la marque de fin de ligne est enlevée ;
- ligne 44 : les composantes (marié, enfants, salaire) de la ligne sont récupérées ;
- ligne 46 : l'impôt est calculé. Le résultat est rendu sous la forme d'un tableau associatif (ligne 76) ;
- ligne 48 : au tableau récupéré précédemment, on rajoute les clés **[marié, enfants, salaire]** ;
- ligne 49 : le résultat est mémorisé dans le fichier des résultats sous la forme d'une chaîne jSON ;
- lignes 53-54 : une fois le fichier des données contribuables exploité totalement, les fichiers sont fermés ;
- ligne 60 : la fonction qui supprime la marque de fin de ligne d'une ligne $*ligne*. La marque de fin de ligne est la chaîne "\r\n" sur les systèmes windows, "\n" sur les systèmes Unix. Le résultat est la chaîne d'entrée sans sa marque de fin de ligne.
- lignes 63-64 : *substr($chaîne,$début,$taille)* est la sous-chaîne de $*chaîne* commençant au caractère $*début* et ayant au plus $*taille* caractères ;

La fonction **[calculImpot]** est la suivante :

```
// constantes globales
define("PLAFOND_QF_DEMI_PART", 1551);
```

```
1.  // calcul de l'impôt
2.  // ----------------------------------------------------------------------
3.  function calculImpot(string $marié, int $enfants, float $salaire, array $limites, array $coeffR, array $coe
    ffN): array {
4.    // $marié : oui, non
5.    // $enfants : nombre d'enfants
6.    // $salaire : salaire annuel
7.    // $limites, $coeffR, $coeffN : les tableaux des données permettant le calcul de l'impôt
8.    //
9.    // calcul de l'impôt avec enfants
10.   $result1 = calculImpot2($marié, $enfants, $salaire, $limites, $coeffR, $coeffN);
11.   $impot1 = $result1["impôt"];
12.   // calcul de l'impôt sans les enfants
13.   if ($enfants != 0) {
14.     $result2 = calculImpot2($marié, 0, $salaire, $limites, $coeffR, $coeffN);
15.     $impot2 = $result2["impôt"];
16.     // application du plafonnement du quotient familial
17.     if ($enfants < 3) {
18.       // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants
19.       $impot2 = $impot2 - $enfants * PLAFOND_QF_DEMI_PART;
20.     } else {
21.       // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants, le double pour les suivants
22.       $impot2 = $impot2 - 2 * PLAFOND_QF_DEMI_PART - ($enfants - 2) * 2 * PLAFOND_QF_DEMI_PART;
23.     }
24.   } else {
25.     $impot2 = $impot1;
26.     $result2 = $result1;
27.   }
28.   // on prend l'impôt le plus fort avec le taux et la surcôte qui vont avec
29.   if ($impot1 > $impot2) {
30.     $impot = $impot1;
31.     $taux = $result1["taux"];
```

```
32.     $surcôte = $result1["surcôte"];
33.   } else {
34.     $surcôte = $impot2 - $impot1 + $result2["surcôte"];
35.     $impot = $impot2;
36.     $taux = $result2["taux"];
37.   }
38.   // calcul d'une éventuelle décôte
39.   $décôte = getDecote($marié, $salaire, $impot);
40.   $impot -= $décôte;
41.   // calcul d'une éventuelle réduction d'impôts
42.   $réduction = getRéduction($marié, $salaire, $enfants, $impot);
43.   $impot -= $réduction;
44.   // résultat
45.   return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" => $réduction,
    "taux" => $taux];
46. }
```

Commentaires

- ligne 10 : l'impôt brut est calculé avec les enfants. On obtient un résultat sous la forme **["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]**] avec :
  - **['impôt']** : l'impôt brut ;
  - **['surcôte']** : le montant de la surcôte s'il y a. Celle-ci existe lorsque l'abattement de 10 % dépasse le seuil de 12502 euros ;
  - **['taux']** : le taux d'imposition du contribuable ;
- ligne 11 : l'impôt **[impot1]** brut à payer ;
- lignes 13-14 : si le contribuable a au moins un enfant, le calcul de l'impôt est refait avec les mêmes données mais avec 0 enfant. Ce second calcul est nécessaire pour voir si la réduction amenée par les enfants (nbParts*coeffN) est supérieure à un certain seuil ;
- ligne 15 : l'impôt brut **[impot2]** à payer ;
- lignes 16-23 : pour l'impôt brut **[impot2]**, on fait jouer maintenant les enfants : chaque 1/2 part amenée par les enfants permet une réduction de **[PLAFOND_QF_DEMI_PART]** euros ;
- lignes 25-26 : cas où le contribuable n'a pas d'enfants. Dans ce cas, le calcul de **[impot2]** est inutile. Il est égal à **[impot1]** ;
- lignes 29-37 : deux impôts bruts ont été calculés **[impot1, impot2]**. L'administration fiscale retient le plus fort des deux. On obtient un impôt brut **[impot]** ;
- lignes 39-40 : le montant brut **[impot]** peut subir une décôte ;
- lignes 42-43 : le montant brut **[impot]** peut subir une réduction ;
- ligne 45 : **[impot]** est désormais l'impôt net à payer. On rend les résultats ;

La fonction **[calculImpot2]** est la suivante :

```
1.
2.  function calculImpot2(string $marié, int $enfants, float $salaire, array $limites, array $coeffR, array $co
    effN): array {
3.    // $marié : oui, non
4.    // $enfants : nombre d'enfants
5.    // $salaire : salaire annuel
6.    // $limites, $coeffR, $coeffN : les tableaux des données permettant le calcul de l'impôt
7.    //
8.    // nombre de parts
9.    $marié = strtolower($marié);
10.   if ($marié === "oui") {
11.     $nbParts = $enfants / 2 + 2;
12.   } else {
13.     $nbParts = $enfants / 2 + 1;
14.   }
15.   // 1 part par enfant à partir du 3ième
16.   if ($enfants >= 3) {
17.     // une demi-part de + pour chaque enfant à partir du 3ième
18.     $nbParts += 0.5 * ($enfants - 2);
19.   }
20.   // revenu imposable
21.   $revenuImposable = getRevenuImposable($salaire);
22.   // surcôte
23.   $surcôte = floor($revenuImposable - 0.9 * $salaire);
24.   // pour des pbs d'arrondi
25.   if ($surcôte < 0) {
26.     $surcôte = 0;
27.   }
28.   // quotient familial
29.   $quotient = $revenuImposable / $nbParts;
30.   // est mis à la fin du tableau limites pour arrêter la boucle qui suit
31.   $limites[count($limites) - 1] = $quotient;
```

```
32.   // calcul de l'impôt
33.   $i = 0;
34.   while ($quotient > $limites[$i]) {
35.     $i++;
36.   }
37.   // du fait qu'on a placé $quotient à la fin du tableau $limites, la boucle précédente
38.   // ne peut déborder du tableau $limites
39.   // maintenant on peut calculer l'impôt
40.   $impôt = floor($revenuImposable * $coeffR[$i] - $nbParts * $coeffN[$i]);
41.   // résultat
42.   return ["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]];
43. }
```

Commentaires

- on applique ici le calcul de l'impôt dit au barême progressif ;
- ligne 9 : *strtolower($chaîne)* rend $*chaîne* en minuscules ;
- lignes 10-19 : calcul du nombre de parts du contribuable ;
- ligne 21 : on calcule le revenu imposable à l'aide d'une fonction. En effet, on a vu que ce n'est pas toujours 0.9*revenusAnnuels. L'abattement de 10 % est en effet limité à 12502 euros ;
- ligne 23 : calcul de l'éventuelle surcôte si le revenu imposable est supérieur à 0.9*revenusAnnuels ;
- lignes 25-27 : corrige le fait qu'à cause d'erreurs d'arrondis, on a parfois **[$surcôte=-1]** ;
- ligne 29 : le quotient familial ;
- lignes 30-36 : ce quotient permet de trouver la tranche d'imposition du contribuable ;
- ligne 40 : une fois la tranche d'imposition du contribuable trouvée, son impôt brut peut être calculé. La fonction *floor($x)* rend la valeur entière immédiatement inférieure à **[$x]** ;
- ligne 42 : on rend les informations calculées ;

La fonction **[getRevenuImposable]** est la suivante :

```
// constantes globales
define("ABATTEMENT_DIXPOURCENT_MAX", 12502);
define("ABATTEMENT_DIXPOURCENT_MIN", 437);
```

```
1.  // revenuImposable=salaireAnnuel-abattement
2.  // l'abattement a un min et un max
3.  function getRevenuImposable(float $salaire): float {
4.    // abattement de 10% du salaire
5.    $abattement = 0.1 * $salaire;
6.    // cet abattement ne peut dépasser ABATTEMENT_DIXPOURCENT_MAX
7.    if ($abattement > ABATTEMENT_DIXPOURCENT_MAX) {
8.      $abattement = ABATTEMENT_DIXPOURCENT_MAX;
9.    }
10.   // l'abattement ne peut être inférieur à ABATTEMENT_DIXPOURCENT_MIN
11.   if ($abattement < ABATTEMENT_DIXPOURCENT_MIN) {
12.     $abattement = ABATTEMENT_DIXPOURCENT_MIN;
13.   }
14.   // revenu imposable
15.   $revenuImposable = $salaire - $abattement;
16.   // résultat
17.   return floor($revenuImposable);
18. }
```

Commentaires

- ligne 5 : l'abattement normal est de 10 % du salaire annuel ;
- lignes 7-9 : l'abattement ne peut dépasser l'abattement maximal **[ABATTEMENT_DIXPOURCENT_MAX]** ;
- lignes 10-13 : l'abattement ne peut être inférieur à l'abattement minimal **[ABATTEMENT_DIXPOURCENT_MIN]** ;
- ligne 15 : calcul du revenu imposable ;

La fonction **[getDecote]** est la suivante :

```
// constantes globales
define("PLAFOND_DECOTE_CELIBATAIRE", 1196);
define("PLAFOND_DECOTE_COUPLE", 1970);
define("PLAFOND_IMPOT_COUPLE_POUR_DECOTE", 2627);
define("PLAFOND_IMPOT_CELIBATAIRE_POUR_DECOTE", 1595);
```

```
1.  // calcule une décôte éventuelle
2.  function getDecote(string $marié, float $salaire, float $impots): float {
3.    // au départ, une décôt nulle
4.    $décôte = 0;
```

```
5.    // montant maximal d'impôt pour avoir la décôte
6.    $plafondImpôtPourDécôte = $marié === "oui" ? PLAFOND_IMPOT_COUPLE_POUR_DECOTE : PLAFOND_IMPOT_CELIBATAIRE_POUR_DECOTE;
7.    if ($impots < $plafondImpôtPourDécôte) {
8.      // montant maximal de la décôte
9.      $plafondDécôte = $marié === "oui" ? PLAFOND_DECOTE_COUPLE : PLAFOND_DECOTE_CELIBATAIRE;
10.     // décôte théorique
11.     $décôte = $plafondDécôte - 0.75 * $impots;
12.     // la décôte ne peut dépasser le montant de l'ompôt
13.     if ($décôte > $impots) {
14.       $décôte = $impots;
15.     }
16.     // pas de décôte <0
17.     if ($décôte < 0) {
18.       $décôte = 0;
19.     }
20.   }
21.   // résultat
22.   return ceil($décôte);
23. }
```

Commentaires

- ligne 6 : montant maximal de l'impôt brut pour avoir droit à une décôte. Ce montant est différent pour les célibataires et les couples ;
- ligne 7 : si le contribuable a droit à la décôte ;
- ligne 11 : la formule de la décôte. **[plafondDécôte]** est le montant maximal de la décôte. Ce montant maximal est calculé ligne 9. Là encore il dépend de la situation du contribuable, marié ou célibataire ;
- lignes 13-15 : la décôte ne peut être supérieure à l'impôt brut à payer. C'est le cas par exemple si **[impots]** vaut 0 en ligne 11 ;
- lignes 17-19 : pour éviter un arrondi à -1 ;

La fonction **[getRéduction]** est la suivante :

```
// constantes globales
define("PLAFOND_REVENUS_CELIBATAIRE_POUR_REDUCTION", 21037);
define("PLAFOND_REVENUS_COUPLE_POUR_REDUCTION", 42074);
define("VALEUR_REDUC_DEMI_PART", 3797);
```

```
1.  // calcule une réduction éventuelle
2.  function getRéduction(string $marié, float $salaire, int $enfants, float $impots): float {
3.    // le plafond des revenus pour avoir droit à la réduction de 20%
4.    $plafondRevenuPourRéduction = $marié === "oui" ? PLAFOND_REVENUS_COUPLE_POUR_REDUCTION : PLAFOND_REVENUS_
    CELIBATAIRE_POUR_REDUCTION;
5.    $plafondRevenuPourRéduction += $enfants * VALEUR_REDUC_DEMI_PART;
6.    if ($enfants > 2) {
7.      $plafondRevenuPourRéduction += ($enfants - 2) * VALEUR_REDUC_DEMI_PART;
8.    }
9.    // revenu imposable
10.   $revenuImposable = getRevenuImposable($salaire);
11.   // réduction
12.   $réduction = 0;
13.   if ($revenuImposable < $plafondRevenuPourRéduction) {
14.     // réduction de 20%
15.     $réduction = 0.2 * $impots;
16.   }
17.   // résultat
18.   return ceil($réduction);
19. }
```

Commentaires

- lignes 4-10 : pour avoir droit à une réduction d'impôt, il faut que le revenu imposable (ligne 10) soit inférieur à un plafond calculé lignes 4-8 ;
- lignes 13-16 : s'il remplit les conditions, le contribuable a droit à une réduction d'impôt de 20 % (ligne 15) ;

1.4.3.2 Résultats

Le fichier des données *taxpayersdata.txt* (marié, enfants, salaire) :

```
1   oui,2,55555
2   oui,2,50000
3   oui,3,50000
4   non,2,100000
5   non,3,100000
6   oui,3,100000
7   oui,5,100000
```

```
8   non,0,100000
9   oui,2,30000
10  non,0,200000
11  oui,3,200000
```

Les fichier *résultats.txt* (marié, enfants, salaire, impôt, surcôte, décôte, réduction, taux d'imposition) des résultats obtenus :

```
1   {"marié":"oui","enfants":"2","salaire":"55555","impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.1
    4}
2   {"marié":"oui","enfants":"2","salaire":"50000","impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux"
    :0.14}
3   {"marié":"oui","enfants":"3","salaire":"50000","impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14
    }
4   {"marié":"non","enfants":"2","salaire":"100000","impôt":19884,"surcôte":4480,"décôte":0,"réduction":0,"taux
    ":0.41}
5   {"marié":"non","enfants":"3","salaire":"100000","impôt":16782,"surcôte":7176,"décôte":0,"réduction":0,"taux
    ":0.41}
6   {"marié":"oui","enfants":"3","salaire":"100000","impôt":9200,"surcôte":2180,"décôte":0,"réduction":0,"taux"
    :0.3}
7   {"marié":"oui","enfants":"5","salaire":"100000","impôt":4230,"surcôte":0,"décôte":0,"réduction":0,"taux":0.
    14}
8   {"marié":"non","enfants":"0","salaire":"100000","impôt":22986,"surcôte":0,"décôte":0,"réduction":0,"taux":0
    .41}
9   {"marié":"oui","enfants":"2","salaire":"30000","impôt":0,"surcôte":0,"décôte":0,"réduction":0,"taux":0}
10  {"marié":"non","enfants":"0","salaire":"200000","impôt":64210,"surcôte":7498,"décôte":0,"réduction":0,"taux
    ":0.45}
11  {"marié":"oui","enfants":"3","salaire":"200000","impôt":42842,"surcôte":17283,"décôte":0,"réduction":0,"tau
    x":0.41}
```

Les résultats obtenus sont conformes à ceux obtenus avec le simulateur de l'administration fiscale.

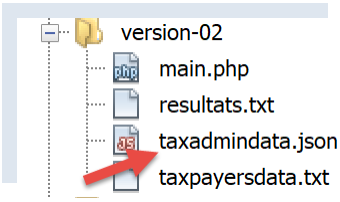
1.4.3.3 Conclusion

L'algorithme de calcul de l'impôt, même dans des cas réputés simples, est complexe. Nous ne reviendrons plus dessus. Au fil des versions, son cœur restera le même malgré quelques changements de présentation. On ne commentera alors que ces derniers.

1.4.4 Version 2

1.4.4.1 Les modifications

Dans la version précédente, les données nécessaires au calcul de l'impôt étaient codées en dur sous la forme de constantes et de tableaux. Cette méthode est à prohiber. Dans la nouvelle version, ces données sont externalisées dans un fichier jSON :



Le contenu du fichier **[taxadmindata.json]** est le suivant :

```
1   {
2       "limites": [9964, 27519, 73779, 156244, 0],
3       "coeffR": [0, 0.14, 0.3, 0.41, 0.45],
4       "coeffN": [0, 1394.96, 5798, 13913.69, 20163.45],
5       "PLAFOND_QF_DEMI_PART": 1551,
6       "PLAFOND_REVENUS_CELIBATAIRE_POUR_REDUCTION": 21037,
7       "PLAFOND_REVENUS_COUPLE_POUR_REDUCTION": 42074,
8       "VALEUR_REDUC_DEMI_PART": 3797,
9       "PLAFOND_DECOTE_CELIBATAIRE": 1196,
10      "PLAFOND_DECOTE_COUPLE": 1970,
11      "PLAFOND_IMPOT_COUPLE_POUR_DECOTE": 2627,
12      "PLAFOND_IMPOT_CELIBATAIRE_POUR_DECOTE": 1595,
13      "ABATTEMENT_DIXPOURCENT_MAX": 12502,
14      "ABATTEMENT_DIXPOURCENT_MIN": 437
15  }
```

La nouvelle version **[version-02/main.php]** est la suivante :

```
1.  <?php
2.
```

```php
// respect strict des types déclarés des paramètres des fonctions
declare (strict_types=1);

// définition des constantes
$TAXPAYERSDATA = "taxpayersdata.txt";
$RESULTATS = "resultats.txt";
$TAXADMINDATA = "taxadmindata.json";

// on récupère le contenu du fichier des données fiscales
$fileContents = \file_get_contents($TAXADMINDATA);
$erreur = FALSE;
// erreur ?
if (!$fileContents) {
  // on note l'erreur
  $erreur = TRUE;
  $message = "Le fichier des données [$TAXADMINDATA] n'existe pas";
}

if (!$erreur) {
  // on récupère le code JSON du fichier de configuration dans un tableau associatif
  $taxAdminData = \json_decode($fileContents, true);
  // erreur ?
  if (!$taxAdminData) {
    // on note l'erreur
    $erreur = TRUE;
    $message = "Le fichier de données JSON [$TAXADMINDATA] n'a pu être exploité correctement";
  }
}

// erreur ?
if ($erreur) {
  // affichage d'un msg d'erreur puis arrêt
  print "$message\n";
  exit;
}

// ouverture fichier des résultats
$résultats = fopen($RESULTATS, "w");
if (!$résultats) {
  print "Impossible de créer le fichier des résultats [$RESULTATS]\n";
  // sortie
  exit;
}

// ouverture fichier des données contribuables
$taxpayersdata = fopen($TAXPAYERSDATA, "r");
if (!$taxpayersdata) {
  print "Impossible d'ouvrir le fichier des contribuables [$TAXPAYERSDATA]\n";
  // sortie
  exit;
}

// on exploite la ligne courante du fichier des données
while ($ligne = fgets($taxpayersdata, 100)) {
  // on enlève l'éventuelle marque de fin de ligne
  $ligne = cutNewLineChar($ligne);
  // on récupère les 3 champs marié:enfants:salaire qui forment $ligne
  list($marié, $enfants, $salaire) = explode(",", $ligne);
  // on calcule l'impôt
  $result = calculImpot($taxAdminData, $marié, (int) $enfants, (int) $salaire);
  // on inscrit le résultat dans le fichier des résultats
  $result = ["marié" => $marié, "enfants" => $enfants, "salaire" => $salaire] + $result;
  fputs($résultats, \json_encode($result, JSON_UNESCAPED_UNICODE) . "\n");
  // donnée suivante
}
// on ferme les fichiers
fclose($taxpayersdata);
fclose($résultats);

// fin
exit;

// ----------------------------------------------------------------------------
function cutNewLinechar(string $ligne) {
  …
  // fin
  return($ligne);
}

// calcul de l'impôt
// ----------------------------------------------------------------------------
function calculImpot(array $taxAdminData, string $marié, int $enfants, float $salaire) {
  // $marié : oui, non
  // $enfants : nombre d'enfants
  // $salaire : salaire annuel
  // $taxAdminData : données de l'administration fiscale
  //
  // calcul de l'impôt avec enfants
```

```
91.    $result1 = calculImpot2($taxAdminData, $marié, $enfants, $salaire);
92.    $impot1 = $result1["impôt"];
93.    // calcul de l'impôt sans les enfants
94.    if ($enfants != 0) {
95.      $result2 = calculImpot2($taxAdminData, $marié, 0, $salaire);
96.      $impot2 = $result2["impôt"];
97.      // application du plafonnement du quotient familial
98.      if ($enfants < 3) {
99.        // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants
100.       $impot2 = $impot2 - $enfants * $taxAdminData["PLAFOND_QF_DEMI_PART"];
101.     } else {
102.       // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants, le double pour les suivants
103.       $impot2 = $impot2 - 2 * $taxAdminData["PLAFOND_QF_DEMI_PART"] - ($enfants - 2) * 2 * $taxAdminData["PLAFOND_QF_
    DEMI_PART"];
104.     }
105.   } else {
106.     $impot2 = $impot1;
107.     $result2 = $result1;
108.   }
109.   // on prend l'impôt le plus fort avec le taux et la surcôte qui vont avec
110.   if ($impot1 > $impot2) {
111.     $impot = $impot1;
112.     $taux = $result1["taux"];
113.     $surcôte = $result1["surcôte"];
114.   } else {
115.     $surcôte = $impot2 - $impot1 + $result2["surcôte"];
116.     $impot = $impot2;
117.     $taux = $result2["taux"];
118.   }
119.   // calcul d'une éventuelle décôte
120.   $décôte = getDecote($taxAdminData, $marié, $salaire, $impot);
121.   $impot -= $décôte;
122.   // calcul d'une éventuelle réduction d'impôts
123.   $réduction = getRéduction($taxAdminData, $marié, $salaire, $enfants, $impot);
124.   $impot -= $réduction;
125.   // résultat
126.   return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" => $réduction, "
    taux" => $taux];
127. }
128.
129. // ----------------------------------------------------------------------
130. function calculImpot2(array $taxAdminData, string $marié, int $enfants, float $salaire) {
131.   // $marié : oui, non
132.   …
133.   // résultat
134.   return ["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]];
135. }
136.
137. // revenuImposable=salaireAnnuel-abattement
138. // l'abattement a un min et un max
139. function getRevenuImposable(array $taxAdminData, float $salaire): float {
140.   …
141.   // résultat
142.   return floor($revenuImposable);
143. }
144.
145. // calcule une décôte éventuelle
146. function getDecote(array $taxAdminData, string $marié, float $salaire, float $impots): float {
147.   …
148.   // résultat
149.   return ceil($décôte);
150. }
151.
152. // calcule une réduction éventuelle
153. function getRéduction(array $taxAdminData, string $marié, float $salaire, int $enfants, float $impots): float {
154.   …
155.   // résultat
156.   return ceil($réduction);
157. }
```

Commentaires

- lignes 11-19 : on essaie de lire le contenu du fichier jSON nommé **[TAXADMINDATA]** ;
- lignes 21-30 : si on a réussi à lire le fichier jSON, son contenu est décodé dans le tableau associatif **[$taxAdminData]** ;
- lignes 32-37 : si on a rencontré une erreur dans une des deux opérations précédentes, on écrit un message d'erreur sur la console et on s'arrête ;
- la différence avec la version 01 est qu'ici les données (tableaux et constantes) de l'administration fiscale sont dans le tableau associatif **[$taxAdminData]** alors que dans la version 01, elles étaient dans des tableaux et constantes globales. C'est la globalité de ces constantes qui fait la différence entre les deux versions :
  - dans la version 01, les constantes étaient connues dans toutes les fonction de **[main.php]** ;
  - pour arriver au même résultat dans la version 02, il faut passer le tableau associatif **[$taxAdminData]** en paramètre à toutes les fonctions (lignes 84, 130, 139, 146, 153) ;

- chaque fonction de la version 02 doit utiliser le contenu du tableau **[$taxAdminData]** ;
- lignes 84-127 : dans la fonction **[calculerImpot]**, là où on utilisait des constantes globales ou les tableaux **[limites, coeffR, coeffN]**, on utilise désormais le contenu du tableau **[$taxAdminData]** reçu en paramètre (lignes 100, 101) ;
- toutes les autres fonctions sont réécrites de la même façon ;

Les résultats obtenus sont les mêmes que ceux obtenus dans la version précédente.

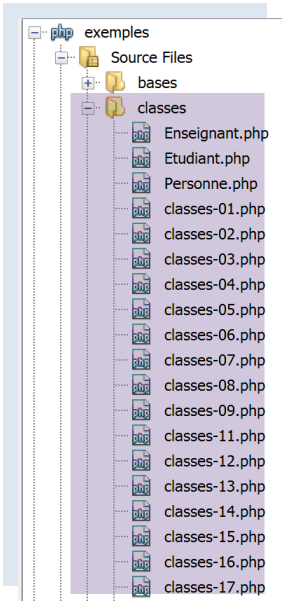
1.4.4.2 Conclusion

La version 02 est bien plus souple que la version 01. En 2020, l'algorithme de calcul de l'impôt sera probablement le même qu'en 2019. Seules les tranches d'imposition et les constantes de calcul auront changé. Il suffira alors de mettre à jour le fichier **[taxadmindata.json]**. Avec la version 01, il faut aller dans le code modifier les tranches d'imposition et les constantes de calcul. Or il est probable que les gens qui ont à changer les valeurs des tranches d'imposition et des constantes de calcul n'ont pas accès au code de l'algorithme.

1.5 Les classes

Vocabulaire : Une classe **class** est un type PHP. Une variable de ce type est appelée **objet**. Un objet est une **instance** (exemplaire) de classe.

1.5.1 L'arborescence des scripts



1.5.2 Toute variable peut devenir un objet doté d'attributs

Le script **[classes-01.php]** est le suivant :

```
1.  <?php
2.
3.  // un objet générique
4.  // $obj1=new stdClass();
5.  // toute variable peut avoir des attributs par construction
6.  $obj1->attr1 = "un";
7.  $obj1->attr2 = 100;
8.  // affiche l'objet
9.  print "objet1=[$obj1->attr1,$obj1->attr2]\n";
10. // modifie l'objet
11. $obj1->attr2 += 100;
12. // affiche l'objet
13. print "objet1=[$obj1->attr1,$obj1->attr2]\n";
14. // copie la valeur de objet1 (adresse de l'objet pointé) à objet2
15. // les deux variables sont alors différentes mais pointent sur le même objet
16. $obj2 = $obj1;
17. // modifie obj2
18. $obj2->attr2 = 0;
19. // affiche les deux objets
20. print "objet1=[$obj1->attr1,$obj1->attr2]\n";
```

```php
print "objet2=[$obj2->attr1,$obj2->attr2]\n";
// change l'objet pointé par obj1
$obj1 = new stdClass();
print "obj1 :\n";
print_r($obj1);
print "obj2 :\n";
print_r($obj2);
// affecte la référence (l'adresse) de objet2 à objet3
// $obj2 et $obj3 sont alors une seule et même variable
$obj3 = &$obj2;
print "obj2 :\n";
print_r($obj2);
print "obj3 :\n";
print_r($obj3);
// modifie obj3
$obj3->attr2 = 10;
// affiche les deux objets
print "objet2=[$obj2->attr1,$obj2->attr2]\n";
print "objet3=[$obj3->attr1,$obj3->attr2]\n";
// change l'objet pointé par obj2
$obj2 = new stdClass();
$obj2->attr3 = "deux";
$obj2->attr4 = 20;
// affiche les deux objets $obj2 et $obj3
print "obj2 :\n";
print_r($obj2);
print "obj3 :\n";
print_r($obj3);

// un objet est-il un dictionnaire ?
print count($obj3) . "\n";
while (list($attribut, $valeur) = each($obj3)) {
  print "obj3[$attribut]=$valeur\n";
}
// fin
exit;
```

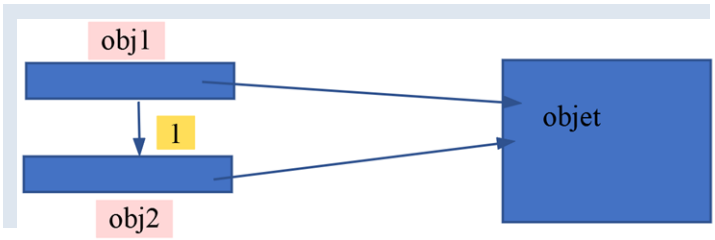
Résultats :

```
Warning: Creating default object from empty value in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_14.php on line 6
objet1=[un,100]
objet1=[un,200]
objet1=[un,0]
objet2=[un,0]
obj1 :
stdClass Object
(
)
obj2 :
stdClass Object
(
    [attr1] => un
    [attr2] => 0
)
obj2 :
stdClass Object
(
    [attr1] => un
    [attr2] => 0
)
obj3 :
stdClass Object
(
    [attr1] => un
    [attr2] => 0
)
objet2=[un,10]
objet3=[un,10]
obj2 :
stdClass Object
(
    [attr3] => deux
    [attr4] => 20
)
obj3 :
stdClass Object
```

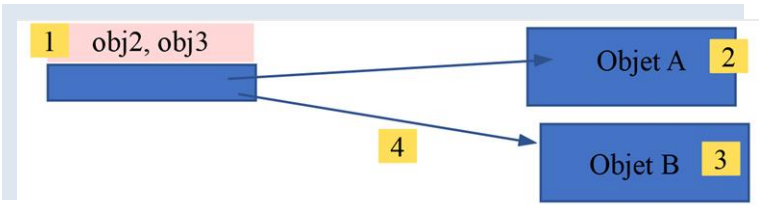
```
38  (
39      [attr3] => deux
40      [attr4] => 20
41  )
42
43  Warning: count(): Parameter must be an array or an object that implements Countable in C:\Data\st-
    2019\dev\php7\php5-exemples\exemples\exemple_14.php on line 50
44  1
45
46  Deprecated: The each() function is deprecated. This message will be suppressed on further calls in
    C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_14.php on line 51
47  obj3[attr3]=deux
48  obj3[attr4]=20
```

Commentaires

- ligne 6 : la notation *$obj->attr* désigne l'attribut *attr* de la variable $*obj*. S'il n'existe pas il est créé, faisant ainsi de la variable $*obj*, un objet avec attributs. Nous avons vu que PHP crée alors par défaut un objet de type **stdClass** ;
- ligne 16 : l'expression *$obj2=$obj1*, lorsque $*obj1* est un objet, est une copie d'objets par **référence** : $*obj2* et $*obj1* sont des références (adresses) sur un même objet. L'objet lui-même peut être modifié par l'une ou l'autre des références ;
- lignes 23-27 : visent à montrer que $*obj1* et $*obj2* sont deux variables différentes : elles ne sont pas à la même adresse mémoire :
  - *$obj2=$obj1* a copié la valeur de $*obj1* dans la variable $*obj2* (opération 1 ci-dessus). La valeur de $*obj1* est l'adresse d'un l'objet. Ainsi $*obj1* et $*obj2* pointent sur le même objet. Lorsqu'on manipule une variable $*obj* et que celle-ci pointe sur un objet, PHP manipule l'objet pointé par la variable $*obj*. D'après le schéma ci-dessous, on voit qu'on peut modifier l'objet pointé soit via $*obj1* soit via $*obj2*. C'est ce que montrent les lignes 4 et 5 des résultats ;



- ligne 30 : l'expression *$obj3=&$obj2* fait que *$obj2 et $obj3* sont à la même adresse **`[1 ci-dessous]`**. On pourrait dire que les deux variables sont des alias du même emplacement mémoire. Elles pointent toutes les deux sur un objet, **Objet A** ci-dessous **`[2]`** ;
  - l'opération *$obj2=new stdClass()* fait qu'un nouvel objet, **Objet B** est créé **`[3 ci-dessous]`** et l'adresse de ce nouvel objet est affectée à la variable $*obj2*. Puisque $*obj2* et $*obj3* sont deux alias du même emplacement mémoire, $*obj3* pointe également sur le nouvel objet **Objet B**. C'est ce que montrent les lignes 16-27 et 30-41 des résultats ;



- lignes 52-54 : montrent qu'un objet peut être parcouru comme un dictionnaire. Les clés du dictionnaire sont les noms des attributs et les valeurs du dictionnaire, les valeurs de ces mêmes attributs ;
- ligne 51 : la fonction *count* peut être appliquée à un objet (moyennant un warning) mais ne donne pas, comme on aurait pu s'y attendre, le nombre d'attributs. Donc un objet présente des similitudes avec un dictionnaire mais n'en est pas un ;

1.5.3 Une classe Personne sans attributs déclarés

Le script **`[classes-02.php]`** est le suivant :

```
1.  <?php
2.
3.  class Personne {
4.
5.    // attributs de la classe
6.    // non déclarés - peuvent être créés dynamiquement
7.    // méthode
8.    function identite() {
```

```
9.      // a priori, utilise des attributs inexistants
10.     return "[$this->prenom,$this->nom,$this->age]";
11.   }
12.
13. }
14.
15. // test
16. // les attributs sont publics et peuvent être créés dynamiquement
17. $p = new Personne();
18. $p->prenom = "Paul";
19. $p->nom = "Langevin";
20. $p->age = 48;
21. // appel d'une méthode
22. print "personne=" . $p->identite() . "\n";
23. // fin
24. exit;
```

Résultats :

```
1   personne=[Paul,Langevin,48]
```

Commentaires

- lignes 3-13 : définissent une **classe** *Personne*. Une classe est un moule à partir duquel on crée des objets. Elle regroupe des **attributs** et des fonctions appelées **méthodes**. Il n'y a pas obligation à déclarer les attributs ;
- lignes 8-11 : la méthode *identité* affiche la valeur de trois attributs non déclarés dans la classe. Le mot clé $**this** désigne l'objet auquel on applique la méthode ;
- ligne 17 : on crée un objet $*p* de type *Personne*. Le mot clé **new** sert à créer un nouvel objet. L'opération rend une **référence** sur l'objet créé (donc une adresse). Diverses écritures sont possibles : *new Personne()*, *new Personne*, *new personne*. Le nom de la classe est insensible à la casse ;
- lignes 18-20 : les trois attributs nécessaires à la méthode *identité* sont créés dans l'objet $p ;
- ligne 22 : la méthode *identité* de la classe *Personne* est appliquée sur l'objet $*p*. Dans le code (lignes 8-11) de la méthode *identité*, $*this* référence le même objet que $*p ;*

1.5.4 La classe Personne avec attributs déclarés

Le script **[classes-03.php]** est le suivant :

```
1.  <?php
2.
3.  class Personne {
4.
5.    // attributs de la classe
6.    var $prenom;
7.    var $nom;
8.    var $age;
9.
10.   // méthode
11.   function identite() {
12.     return "[$this->prenom,$this->nom,$this->age]";
13.   }
14.
15. }
16.
17. // test
18. // les attributs sont publics
19. $p = new Personne();
20. $p->prenom = "Paul";
21. $p->nom = "Langevin";
22. $p->age = 48;
23. // appel d'une méthode
24. print "personne=" . $p->identite() . "\n";
25. // fin
26. exit;
```

Résultats :

```
1   personne=[Paul,Langevin,48]
```

Commentaires

- lignes 6-8 : les attributs de la classe sont explicitement déclarés avec le mot clé **var ;**

1.5.5 La classe Personne avec un constructeur

Les exemples précédents montraient des classes *Personne* exotiques telles qu'on pouvait les trouver dans PHP 4. Il n'est pas conseillé de suivre ces exemples. Nous présentons maintenant une classe *Personne* **[classes-04.php]** correspondant aux bonnes pratiques de PHP 7 :

```
1.  <?php
2.
3.  class Personne {
4.  // attributs de la classe
5.    private $prenom;
6.    private $nom;
7.    private $age;
8.
9.  // getters and setters
10.   public function getPrenom(): string {
11.     return $this->prenom;
12.   }
13.
14.   public function getNom(): string {
15.     return $this->nom;
16.   }
17.
18.   public function getAge(): int {
19.     return $this->age;
20.   }
21.
22.   public function setPrenom(string $prenom): void {
23.     $this->prenom = $prenom;
24.   }
25.
26.   public function setNom(string $nom): void {
27.     $this->nom = $nom;
28.   }
29.
30.   public function setAge(int $age): void {
31.     $this->age = $age;
32.   }
33.
34. // constructeur
35.   public function __construct(string $prenom, string $nom, int $age) {
36.     // on passe par les set
37.     $this->setPrenom($prenom);
38.     $this->setNom($nom);
39.     $this->setAge($age);
40.   }
41.
42. // méthode toString
43.   public function __toString(): string {
44.     return "[$this->prenom,$this->nom,$this->age]";
45.   }
46.
47. }
48.
49. // test
50. // création d'un objet personne
51. $p = new Personne("Paul", "Langevin", 48);
52. // identité de cette personne
53. print "personne=$p\n";
54. // on change l'âge
55. $p->setAge(14);
56. // identité de la personne
57. print "personne=$p\n";
58. // fin
59. exit;
```

Résultats :

```
1   personne=[Paul,Langevin,48]
2   personne=[Paul,Langevin,14]
```

Commentaires

- lignes 3-47 : la classe *Personne ;*
- lignes 5-7 : les attributs privés (**private**) de la classe. Ces attributs ne sont visibles qu'à l'intérieur de la classe. Les autres mots clés utilisables sont :
  - **public** : fait de l'attribut un attribut *public* visible de l'extérieur de la classe,
  - **protected** : fait de l'attribut un attribut *protégé* visible de l'intérieur de la classe et des classes dérivées de celle-ci ;
- parce que les attributs sont privés, on ne peut y accéder de l'extérieur de la classe. On ne peut donc écrire le code suivant :

```
1   $p=new Personne() ;
2   $p->nom="Landru" ;
```

Ici, on est en-dehors de la classe *Personne*. Comme l'attribut *nom* est privé, la ligne 2 est incorrecte. Pour initialiser les champs privés de l'objet $p, il y a deux moyens :

- utiliser les méthodes publiques **set** et **get** (le nom de ces méthodes peut être quelconque) des lignes 10-32. On pourra alors écrire :

```
1   $p=new Personne() ;
2   $p->setNom("Landru") ;
```

- utiliser le constructeur des lignes 35-40. On écrira alors :

```
1   $p=new Personne("Michel","Landru",44) ;
```

L'écriture ci-dessus, appelle automatiquement la méthode de la classe *Personne* appelée *__construct ;*

- ligne 53 : cette ligne affiche la personne $*p* sous la forme d'une chaîne de caractères. Pour ce faire, la méthode de la classe *Personne* appelée *__toString* (lignes 43-45) est utilisée ;
- toutes les méthodes de la classe (fonctions) ont été préfixées par le mot clé **public** qui indique que la fonction est visible en-dehors de la classe. Les autres mots clés utilisables sont, comme pour les attributs et avec la même signification, **private** et **protected.** Sans attribut explicite de visibilité, la fonction est de visibilité implicite **public** ;

1.5.6 La classe Personne avec contrôles de validité dans le constructeur

Le constructeur d'une classe est le bon endroit pour vérifier que les valeurs d'initialisation de l'objet sont corrects. Mais un objet peut être également initialisé par ses méthodes *set* ou équivalentes. Pour éviter de mettre à deux endroits différents les mêmes vérifications, on pourra mettre ces dernières dans les méthodes *set*. Si une valeur d'initialisation d'un objet se révèle incorrecte, on lancera une **exception**. Voici un exemple.

On déplace tout d'abord la définition de la classe *Personne* dans son propre fichier **[Personne.php]** :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms;
7.  namespace Exemples;
8.
9.  // classe Personne
10. class Personne {
11. // attributs de la classe
12.   private $prenom;
13.   private $nom;
14.   private $age;
15.
16. // getters et setters
17.   public function getPrenom(): string {
18.     return $this->prenom;
19.   }
20.
21.   public function getNom(): string {
22.     return $this->nom;
23.   }
24.
25.   public function getAge(): int {
26.     return $this->age;
27.   }
28.
29.   public function setPrenom(string $prénom): void {
30.     // le prénom doit être non vide
```

```
31.     $prénom = trim($prénom);
32.     if ($prénom === "") {
33.       throw new \Exception("Le prénom doit être non vide");
34.     } else {
35.       $this->prenom = $prénom;
36.     }
37.   }
38.
39.   public function setNom(string $nom): void {
40.     // le nom doit être non vide
41.     $nom = trim($nom);
42.     if ($nom === "") {
43.       throw new \Exception("Le nom doit être non vide");
44.     } else {
45.       $this->nom = $nom;
46.     }
47.   }
48.
49.   public function setAge(int $âge): void {
50.     // l'âge doit être valide
51.     if ($âge < 0) {
52.       throw new \Exception("L'âge doit être un entier positif ou nul");
53.     } else {
54.       $this->age = $âge;
55.     }
56.   }
57.
58. // constructeur
59.   public function __construct(string $prenom, string $nom, int $age) {
60.     // on passe par les set
61.     $this->setPrenom($prenom);
62.     $this->setNom($nom);
63.     $this->setAge($age);
64.   }
65.
66.   // méthode
67.   public function initWithPersonne(Personne $p): void {
68.     // initialise l'objet courant avec une personne $p
69.     $this->__construct($p->prenom, $p->nom, $p->age);
70.   }
71.
72.   // méthode toString
73.   function __toString(): string {
74.     return "[$this->prenom,$this->nom,$this->age]";
75.   }
76.
77. }
```

Commentaires

- ligne 4 : on demande à ce que le type des paramètres des fonctions soit respecté lorsqu'il est déclaré ;
- ligne 7 : définit un espace de noms (**namespace**). Le nom complet (on dit *qualifié*) de la classe *Personne* est alors*Exemples**Personne*. Notez le caractère \ commençant le nom qualifié : on a alors un nom qualifié *absolu*. Si ce caractère est absent, on a un nom qualifié *relatif* (relatif à l'espace de noms courant). Ainsi si deux classes A et B font partie du même espace de noms E, dans le code de la classe A on pourra atteindre la classe B par la notation *relative* **B**. Si la classe A fait partie de l'espace de noms E1 et B de l'espace de noms E2, dans le code de A, B sera atteint par la notation *absolue* **\E2\B**. Définir une classe à l'intérieur d'un espace de noms n'est pas obligatoire mais Netbeans émet un warning si on ne le fait pas. Donc on le fera. Par ailleurs, les espaces de noms devraient correspondre à l'arborescence des fichiers. Ainsi la classe A dans un espace de noms E1 devrait être dans un fichier **E1/A.php**. Ce n'est pas obligatoire mais là encore Netbeans émet un avertissement si on ne le fait pas. Sur l'exemple de la classe **`[\Exemples\Personne]`**, Netbeans émet un avertissement parce que l'arborescence du fichier **`[Personne.php]`** est **`[exemples/classes/Personne.php]`** et ne correspond donc pas à l'espace de noms. Il ne faut pas confondre arborescence et espace de noms. Le nom pleinement qualifié d'une classe utilise un espace de noms et **n'a rien à voir** avec l'arborescence du fichier PHP de la classe). Le lien arborescence / espace de noms est facultatif et peut ne pas être observé, comme nous l'avons fait ici;
- lignes 12-14 : les trois attributs privés de la classe ;
- lignes 29-37 : initialisation de l'attribut *prenom* et vérification de la valeur d'initialisation ;
- ligne 31 : la fonction *trim($chaine)* élimine les espaces qui se trouvent en début et fin de $*chaine*. Ainsi *trim(« abcd »)* est la chaîne «abcd» et *trim « »* est la chaîne vide ;
- ligne 32 : si le prénom est vide, alors on lance une exception (ligne 33) sinon le prénom est mémorisé (ligne 35). Pour lancer une exception on a utilisé ici la classe prédéfinie **`[Exception]`**. On est obligés ici d'utiliser son nom absolu **`[\Exception]`**. Si on utilise son nom relatif **`[Exception]`** alors cette classe sera cherchée dans l'espace de noms du moment, ç-à-d l'espace de noms

[Exemples] de la classe *Personne*. Ainsi l'interpréteur PHP cherchera une classe de nom absolu **[\Exemples\Exception]** qui n'existe pas ;

La classe **[Personne]** est utilisée par le script **[classes-05.php]** suivant :

```
1.  <?php
2.
3.  // respect strict du type des paramètres des fonctions
4.  declare(strict_types=1);
5.  // inclusion définition de la classe Personne
6.  require_once __DIR__ . "/Personne.php";
7.  // pour démonstration
8.  print __DIR__ . "/Personne.php\n";
9.  // nom qualifié de la classe Personne
10. use \Exemples\Personne as Personne;
11.
12. // test
13. // création d'un objet Personne
14. $p = new Personne("Paul", "Langevin", 48);
15. // identité de cette personne
16. print "Exemple1, personne=$p\n";
17. // création d'un objet Personne erroné
18. try {
19.   $p = new Personne("xx", "yy", "zz");
20. } catch (\Exception $e1) {
21.   print "Exemple2, erreur : " . $e1->getMessage() . "\n";
22. } catch (\TypeError $e2) {
23.   print "Exemple2, erreur : " . $e2->getMessage() . "\n";
24. }
25. // création d'un objet Personne erroné
26. try {
27.   $p = new Personne("", "yy", 10);
28. } catch (\Exception $e1) {
29.   print "Exemple3, erreur : " . $e1->getMessage() . "\n";
30. } catch (\TypeError $e2) {
31.   print "Exemple3, erreur : " . $e2->getMessage() . "\n";
32. }
33.
34. // fin
35. exit;
```

Commentaires

- ligne 7 : le script va utiliser la classe **[Personne]**. Il faut alors dire à l'interpréteur PHP où il peut trouver la définition de cette classe. C'est le rôle des instructions **[include fichier]** et **[require fichier]**. Ici on a utilisé l'instruction **[include]**. La différence entre les deux instructions est la suivante : si l'instruction **[include fichier]** rencontre des erreurs lors du chargement de **[fichier]** une erreur de niveau **[E_WARNING]** est émise mais l'exécution continue alors que **[require]** dans le même cas génère une erreur fatale et l'exécution du script s'arrête. Chacune des deux instructions a une variante **[include_once]** et **[require_once]**. Ces deux variantes permettent de gérer le cas des inclusions multiples d'un même fichier. On peut imaginer ici un projet constitué de plusieurs scripts PHP dont plusieurs référencent la classe **[Personne]**. Leur exécution va alors provoquer plusieurs fois l'inclusion du fichier **[Personne.php]** et provoquer une erreur car une classe ne peut être définie deux fois. La solution est d'utiliser les variantes **[_once]** qui assurent que le fichier ne sera inclus qu'une fois dans le script global du projet ;
- ligne 7 : la constante **[__DIR__]** est une constante PHP qui désigne le nom complet du dossier dans lequel se trouve le script contenant la constante **[__DIR__]**. Ainsi l'expression de la ligne 17 :

```
require_once __DIR__."/Personne.php";
```

  sera équivalente à quelque chose comme :

```
require_once 'C:\Data\st-2019\dev\php7\php5-exemples\exemples\classes/Personnes.php'
```

  Dans le chemin du fichier, on peut utiliser indifféremment les signes / et \ ;

- ligne 14 : on utilise la classe **[Personne]** que nous venons de définir. Le script **[classes-05.php]** n'a pas d'espaces de noms. La ligne 14 utilise le nom relatif de la classe **[Personne]** sans espace de noms. En l'absence d'espace de noms de la classe **[Personne]**, celle-ci est cherchée dans le script **[classes-05.php]** lui même et ne sera donc pas trouvée. Il y a deux solutions à ce problème :
  - utiliser le nom complet de la classe **[\Exemples\Personne]** ;
  - utiliser l'instruction **use** de la ligne 10. Celle-ci indique que le code qui suit utilise la classe **[\Exemples\Personne]** ;

- ligne 10 : l'instruction **use** permet à l'interpréteur de savoir que la classe **[Personne]** référencée ligne 14 est en réalité la classe **[\Exemples\Personne]**. Ceci dit, où l'interpréteur va-t-il trouver le code de cette classe ? C'est la ligne 7 qui le lui dit. Celle-ci indique que pour exécuter le script courant il faut également charger le script **[Personne.php]**. On a utilisé ici le nom relatif du fichier. Il sera donc cherché dans le dossier qui contient le script **[classes-05.php]**. Il faut donc que les scripts **[Personne.php]** et **[classes-05.php]** soient dans le même dossier. C'est le cas ici où ils sont tous les deux dans le dossier **[exemples/classes]**. L'instruction de la ligne 10 est équivalente à :

```
use \Exemples\Personne as Personne;
```

  L'instruction **[use]** ci-dessus dit que **l'alias [Personne]** désigne la classe **[\Exemples\Personne]** ;
- ligne 14 : un objet **[Personne]** est créé. C'est la méthode **[__construct]** de la classe **[Personne]** qui va être ici implicitement exécutée ;
- ligne 16 : fait afficher la *Personne $p*. Pour être affichée, la valeur de la variable $p doit être transformée en chaîne de caractères. Implicitement c'est la méthode **[Personne.__toString]** qui est alors exécutée. Celle-ci doit donc rendre une chaîne de caractères ;
- nous avons vu que le constructeur de la classe **[Personne]** pouvait lancer une exception de type **[\Exception]**. Il faut donc gérer celle-ci. Aussi le code de la ligne 14 est-il incomplet. Il faut utiliser celui des lignes 18-24 pour gérer correctement l'exception qui peut se produire. Ici on en produit volontairement une en passant un âge qui n'est pas un entier. Dans ce cas particulier l'exception qui se produit est lancée par l'interpréteur PHP et non par le code de la classe **[Personne]**. En effet, la signature de la méthode **[Personne.__construct]** est la suivante :

```
function __construct(string $prenom, string $nom, int $age)
```

  Il faut donc que le paramètre **[age]** passé au constructeur soit de type **entier**. Si ce n'est pas le cas, l'interpréteur PHP lance une erreur de type **[TypeError]**. Par ailleurs, les méthodes **[set]** de la classe **[Personne]** lancent, elles, une exception de type **[\Exception]**. Comme le constructeur qui les appelle n'a pas de structure *try / catch*, l'exception remonte d'un niveau, au code qui a appelé le constructeur, ç-à-d le code du script **[classes-05.php]**. Finalement, le script **[classes-05.php]** peut recevoir deux types d'exception : **Exception** ou **TypeError.** On notera que lorsque le développeur est sûr que certaines exceptions ne peuvent se produire, il n'utilisera pas les options **catch** correspondantes. Ici elles sont systématiquement utilisées par unique souci de démonstration. Les options **catch** seront cependant utilisées **pour toute exception possible, même peu probable** ;
  Pour cette raison, la structure **try** des lignes 18-24 a deux **catch** pour gérer séparément les deux types d'exception ;
- ligne 20 : on peut écrire indifféremment **[Exception]** ou **[\Exception]** :
  - le 1re version utilise le nom relatif de la classe, relatif à l'espace de noms du script. Celui-ci n'en a pas. Son espace de noms est alors la racine des espaces de noms : \. Donc écrire ici **[Exception]** revient à écrire **[\Exception]**. Or la classe **[Exception]** se trouve bien dans l'espace de noms **[\]** ;

  Il est préférable d'utiliser le nom absolu des exceptions prédéfinies de PHP dans un script n'ayant pas d'espaces de noms lui-même. Ainsi si on décide de donner un espace de noms à ce script, l'écriture des noms absolus de classes reste valide alors que dans l'autre cas, le changement d'espace de noms va provoquer des erreurs sur les noms relatifs des classes ;
- ligne 21 : lorsqu'il y a exception, la méthode **[Exception→getMessage]** permet d'obtenir le message d'erreur de l'exception. Il en est de même pour une erreur de type **[TypeError]**. Dans la méthode **[Personne.setPrenom]**, on a écrit :

```
1      public function setPrenom(string $prénom) {
2        // le prénom doit être non vide
3        $prénom = trim($prénom);
4        if ($prénom === "") {
5          throw new \Exception("Le prénom doit être non vide");
6        } else {
7          $this->prenom = $prénom;
8        }
9    }
```

  Ligne 5, une exception est lancée avec le message d'erreur **[Le prénom doit être non vide]**. C'est ce que récupèrera la méthode **[Exception→getMessage]** de la ligne 29 du script **[classes-05.php]**.

Résultats :

```
1   Exemple1, personne=[Paul,Langevin,48]
2   Exemple2, erreur : Argument 3 passed to Exemples\Personne::__construct() must be of the type integer,
    string given, called in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exemple_18.php on line 19
3   Exemple3, erreur : Le prénom doit être non vide
```

1.5.7 Ajout d'une méthode faisant office de second constructeur

En PHP 7, il n'est pas possible d'avoir plusieurs constructeurs avec des paramètres différents qui permettraient de construire un objet de diverses façons. On peut alors utiliser des méthodes faisant office de constructeur :

```
1.    // méthode
2.    public function initWithPersonne(Personne $p): void {
3.      // initialise l'objet courant avec une personne $p
4.      $this->__construct($p->prenom, $p->nom, $p->age);
5.    }
```

Commentaires

- lignes 2-5 : la méthode *initWithPersonne* permet d'affecter à l'objet courant les valeurs des attributs d'un autre objet *Personne*. Ici, elle fait appel au constructeur *__construct* mais ce n'est pas obligatoire. Elle pourrait initialiser elle-même les attributs de la classe **[Personne]** ;

La nouvelle classe **[Personne]** est utilisée par le script **[classes-06.php]** suivant :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // inclusion définition de la classe Personne
7.  require_once __DIR__ . "/Personne.php";
8.  // déclaration du nom qualifié de la classe Personne
9.  use \Exemples\Personne;
10.
11. // test
12. // création d'un objet Personne
13. try {
14.   $p = new Personne("Paul", "Langevin", 48);
15. } catch (\Exception $e) {
16.   print "erreur : " . $e->getMessage();
17.   exit;
18. }
19. // identité de cette personne
20. print "personne=$p\n";
21. // création d'une seconde personne
22. try {
23.   $p2 = new Personne("Laure", "Adeline", 67);
24. } catch (\Exception $e) {
25.   print "erreur : " . $e->getMessage();
26.   exit;
27. }
28. // initialisation de la première avec les valeurs de la seconde
29. try {
30.   $p->initWithPersonne($p2);
31. } catch (\Exception $e) {
32.   print "erreur : " . $e->getMessage();
33.   exit;
34. }
35.
36. // vérification
37. print "personne=$p\n";
38. // fin
39. exit;
```

- lignes 17, 26, 33 : il est fréquent qu'après une exception, on doive arrêter l'exécution d'un script console si l'erreur rencontrée est irrécupérable. Ce n'est pas le cas dans un script web : on n'arrête pas l'exécution du script mais on fait afficher une page d'erreur. Si on est dans une fonction, ce n'est pas l'instruction **exit** qui sera utilisée mais **return** : on n'arrête pas l'exécution du script (**exit**) mais on sort de la fonction (**return**) après avoir positionné une erreur ;

Résultats :

```
1   personne=[Paul,Langevin,48]
2   personne=[Laure,Adeline,67]
```

1.5.8 Un tableau d'objets [Personne]

L'exemple suivant **[classes-07.php]** montre qu'on peut avoir des tableaux d'objets.

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
```

```
6.  // inclusion définition de la classe [Personne]
7.  require_once __DIR__."./Personne.php";
8.  use \Exemples\Personne;
9.
10. // test
11. // création d'un tableau d'objets Personne
12. // pour faciliter la compréhension du code, on ne gère pas l'éventuelle exception
13. $groupe = [new Personne("Paul", "Langevin", 48), new Personne("Sylvie", "Lefur", 70)];
14.
15. // identité de ces personnes
16. for ($i = 0; $i < count($groupe); $i++) {
17.   print "groupe[$i]=$groupe[$i]\n";
18. }
19.
20. // fin
21. exit;
```

Résultats :

```
1   groupe[0]=[Paul,Langevin,48]
2   groupe[1]=[Sylvie,Lefur,70]
```

Commentaires

- ligne 13 : création d'un tableau de 2 personnes ;
- ligne 16 : parcours du tableau ;
- ligne 17 : *$groupe**[$i]*** est un objet de type *Personne*. La méthode **[Personne.__toString]** est utilisée pour l'afficher ;

1.5.9 Création d'une classe dérivée de la classe Personne

On crée dans un fichier **[Enseignant.php]** la classe **[Enseignant]** suivante :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Exemples;
8.
9.  // une classe dérivée de personne
10. class Enseignant extends Personne {
11.   // attributs
12.   private $discipline;   // discipline enseignée
13.
14.   // getter et setter
15.
16.   public function getDiscipline(): string {
17.     return $this->discipline;
18.   }
19.
20.   public function setDiscipline(string $discipline): void {
21.     $this->discipline = $discipline;
22.   }
23.
24.   // constructeur
25.   public function __construct(string $prénom, string $nom, int $âge, string $discipline) {
26.     // attributs parent
27.     parent::__construct($prénom, $nom, $âge);
28.     // autres attributs
29.     $this->setDiscipline($discipline);
30.   }
31.
32.   // surcharge de la fonction __toString de la classe parente
33.   public function __toString(): string {
34.     return "[" . parent::__toString() . ",$this->discipline]";
35.   }
36.
37. }
```

Commentaires

- ligne 7 : la classe **[Enseignant]** fait partie elle aussi de l'espace de noms **[Exemples]** ;

- ligne 10 : la classe *Enseignant* dérive (**extends**) de la classe *Personne*. La classe dérivée *Enseignant* hérite des attributs et des méthodes de sa classe mère ;
- ligne 12 : la classe *Enseignant* ajoute un nouvel attribut *discipline* qui lui est propre ;
- ligne 25 : le constructeur de la classe *Enseignant* reçoit 4 paramètres :
    - 3 pour initialiser sa classe parent (prénom, nom, âge), ligne 27 ;
    - 1 pour sa propre initialisation (discipline), ligne 29 ;
- ligne 27 : la classe dérivée a accès aux méthodes et constructeurs de sa classe parent via le mot clé **parent ::**. Ici on passe les paramètres (prénom, nom, âge) au constructeur de la classe parent ;
- lignes 33-35 : la méthode *__toString* de la classe dérivée utilise la méthode *__toString* de la classe parent ;

La classe **[Enseignant]** est utilisée par le script **[classes-08.php]** suivant :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // inclusion de la définition des deux classes
7.  require_once __DIR__."/Personne.php";
8.  require_once __DIR__."/Enseignant.php";
9.  // noms qualifiés des deux classes utilisées
10. use \Exemples\Personne;
11. use \Exemples\Enseignant;
12.
13. // test
14. // création d'un tableau d'objets Personne et dérivés
15. // pour la simplicité de l'exemple, on ne gère pas les exceptions
16. $groupe = array(new Enseignant("Paul", "Langevin", 48, "anglais"), new Personne("Sylvie", "Lefur", 70));
17.
18. // identité de ces personnes
19. for ($i = 0; $i < count($groupe); $i++) {
20.   print "groupe[$i]=$groupe[$i]\n";
21. }
22. // fin
23. exit;
```

Commentaires

- lignes 7-8 : il nous faut dire à l'interpréteur PHP où se trouvent les deux classes **[Enseignant, Personne]** ;
- lignes 10-12 : déclaration des noms complets des deux classes. Ceci nous permettra de les désigner dans le code simplement par leur nom sans le suffixe de leurs espaces de noms ;
- ligne 16 : on crée un tableau comportant un type **[Personne]** et un type **[Enseignant]** ;
- lignes 19-22 : affichent les éléments du tableau ;
- ligne 20 : la méthode *__toString* de chaque élément *$groupe**[$i]*** va être appelée. La classe *Personne* a une méthode *__toString*. La classe *Enseignant* en a deux : celle de sa classe parent et la sienne propre. On peut se demander laquelle va être appelée. L'exécution montre que c'est celle de la classe *Enseignant* qui a été appelée. C'est toujours ainsi : lorsqu'une méthode est appelée sur un objet, celle-ci est cherchée dans l'ordre suivant : dans l'objet lui-même, dans sa classe parent s'il en a une, puis dans la classe parent de la classe parent, etc … La recherche s'arrête dès que la méthode a été trouvée.

Résultats :

```
1   groupe[0]=[[Paul,Langevin,48],anglais]
2   groupe[1]=[Sylvie,Lefur,70]
```

1.5.10 Création d'une seconde classe dérivée de la classe Personne

L'exemple suivant crée une classe *Etudiant* dérivée de la classe *Personne*, dans le fichier **[Etudiant.php]** :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Exemples;
8.
9.  class Etudiant extends Personne {
10.   // attributs
11.   private $formation;   // formation suivie
12.
```

```
13.   // getter et setter
14.   public function getFormation(): string {
15.     return $this->formation;
16.   }
17.
18.   public function setFormation(string $formation): void {
19.     $this->formation = $formation;
20.   }
21.
22.   // constructeur
23.   public function __construct(string $prénom, string $nom, int $âge, string $formation) {
24.     // attributs parent
25.     parent::__construct($prénom, $nom, $âge);
26.     // autres attributs
27.     $this->setFormation($formation);
28.   }
29.
30.   // surcharge de la fonction __toString de la classe parente
31.   public function __toString(): string {
32.     return "[" . parent::__toString() . ",$this->formation]]";
33.   }
34.
35. }
```

Cette classe est utilisée par le script **[classes-09.php]** suivant :

```
1   <?php
2
3   // inclusion et définition des classes utilisées par le script
4   require_once __DIR__."/Personne.php";
5   use \Exemples\Personne;
6   require_once __DIR__."/Enseignant.php";
7   use \Exemples\Enseignant;
8   require_once __DIR__."/Etudiant.php";
9   use \Exemples\Etudiant;
10
11  // test
12  // création d'un tableau d'objets personne et dérivés
13  // pour faciliter la compréhension de l'exemple, on ne gère pas les exceptions
14  $groupe = array(new Enseignant("Paul", "Langevin", 48, "anglais"), new Personne("Sylvie", "Lefur", 70), new
    Etudiant("Steve", "Boer", 23, "iup2 qualité"));
15
16  // identité de ces personnes
17  for ($i = 0; $i < count($groupe); $i++) {
18    print "groupe[$i]=$groupe[$i]\n";
19  }
20  // fin
21  exit;
```

Résultats :

```
1   groupe[0]=[[Paul,Langevin,48],anglais]
2   groupe[1]=[Sylvie,Lefur,70]
3   groupe[2]=[[Steve,Boer,23],iup2 qualité]
```

1.5.11 Relation du constructeur d'une classe dérivée avec celui de la classe parent

Dans certains langages orientés objet, le constructeur d'une classe dérivée appelle automatiquement celui de sa classe parent. Le code suivant **[classes-16.php]** montre qu'avec PHP 7 ce n'est pas le cas :

```
1.  <?php
2.
3.  class Classe1 {
4.
5.    // constructeur
6.    public function __construct() {
7.      print "constructeur de la classe Classe1\n";
8.    }
9.
10. }
11.
12. class Classe2 extends Classe1 {
13.
14.   // constructeur
15.   public function __construct() {
```

```
16.     // le constructeur de la classe parent n'est pas appelé implicitement
17.     print "constructeur de la classe Classe2\n";
18.   }
19.
20. }
21.
22. class Classe3 extends Classe1 {
23.
24.   // constructeur
25.   public function __construct() {
26.     // appel explicite du constructeur de la classe parent
27.     parent::__construct();
28.     // code propre à Classe3
29.     print "constructeur de la classe Classe3\n";
30.   }
31.
32. }
33.
34. // tests
35. print "test1---------\n";
36. new Classe2();
37. print "test2---------\n";
38. new Classe3();
```

Résultats

```
1   test1---------
2   constructeur de la classe Classe2
3   test2---------
4   constructeur de la classe Classe1
5   constructeur de la classe Classe3
```

1.5.12 Redéfinition d'une méthode de la classe parent

Nous avons déjà vu qu'une méthode de la classe parent pouvait être redéfinie dans une classe fille. Ainsi la méthode **[__toString]** de la classe **[Personne]** (cf lien) a été redéfinie dans les classes filles **[Enseignant]** (cf. lien) et **[Etudiant]** (cf. lien). Le script **[classes-13.php]** illustre de nouveau le concept :

```
1.  <?php
2.
3.  // respect strict du type des paramètres des fonctions
4.  declare(strict_types=1);
5.
6.  // classe principale
7.  class Classe1 {
8.
9.    public function f(): int {
10.     return 1;
11.   }
12.
13.   function g(): int {
14.     return 2;
15.   }
16.
17. }
18.
19. // classe dérivée
20. class Classe2 extends Classe1 {
21.
22.   // on redéfinit la fonction f de la classe parent
23.   public function f(): int {
24.     return parent::f() + 10;
25.   }
26.
27. }
28.
29. // code
30. $c2 = new Classe2();
31. print $c2->f() . "\n";
32. print $c2->g() . "\n";
33. $c1 = new Classe1();
34. print $c1->f() . "\n";
```

Commentaires

- lignes 7-17 : la classe **[Classe1]** définit deux méthodes **f** et **g** ;
- lignes 20-27 : la classe **[Classe2]** étend la classe **[Classe1]** et redéfinit la méthode **f** de celle-ci ;

Résultats

```
1   11
2   2
3   1
```

Commentaires

- la ligne 30 du code crée un objet $c2 de type **[Classe2]** ;
- la ligne 31 du code appelle la méthode **f** de l'objet $**c2**. Comme celle-ci existe, elle est exécutée ;
- la ligne 32 du code appelle la méthode **g** de l'objet $**c2**. Comme celle-ci n'existe pas, elle est recherchée dans la classe parent où elle est trouvée et exécutée ;
- la ligne 33 du code crée un objet $**c1** de type **[Classe1]** ;
- la ligne 34 du code appelle la méthode **f** de l'objet $**c1**. Comme celle-ci existe, elle est exécutée ;

1.5.13 Passage d'un objet en paramètre d'une fonction

Considérons le script **[classes-14.php]** suivant :

```
1.  <?php
2.
3.  // respect strict du type des paramètres des fonctions
4.  declare(strict_types=1);
5.
6.  // classe principale
7.  class Classe1 {
8.
9.    public function f(): int {
10.     return 1;
11.   }
12.
13.   function g(): int {
14.     return 2;
15.   }
16.
17. }
18.
19. // classe dérivée
20. class Classe2 extends Classe1 {
21.
22.   // on redéfinit la fonction f de la classe parent
23.   public function f(): int {
24.     return parent::f() + 10;
25.   }
26.
27. }
28.
29. // le paramètre de la fonction est de type Classe1 ou dérivé
30. function doSomething(Classe1 $c1): void {
31.   print $c1->f() + $c1->g() . "\n";
32. }
33.
34. // code
35. // on crée un objet de type Classe2 dérivé de Classe1
36. $c2 = new Classe2();
37. // on appelle doSomething avec
38. doSomething($c2);
```

Commentaires

- lignes 7-17 : la classe **[Classe1]** ;
- ligne 20-27 : une classe **[Classe2]** dérivée de **[Classe1]** ;
- ligne 30 : une fonction qui attend un paramètre de type **[Classe1]**. Lorsque le type attendu est une classe alors le paramètre effectif peut être un objet du type attendu **ou dérivé** ;
- lignes 35-38 : on appelle la fonction **[doSomething]** avec un paramètre de type **[Classe2]** alors que c'est le type **[Classe1]** qui est attendu ;

Résultats

```
1   13
```

1.5.14 Classes abstraites

Une classe **abstraite** est une classe incomplète qui ne peut être instanciée. Elle doit obligatoirement être dérivée pour être utilisable.

A quoi sert une classe abstraite ? On a parfois des classes qui partagent une ou des méthodes mais qui se différencient par d'autres méthodes ou d'autres attributs. Il est alors souhaitable de rassembler tout ce qui est commun dans une classe parent. Pour l'instant on n'a pas besoin de classe abstraite. Mais supposons que les classes filles ne diffèrent que par une méthode **M** : la signature de la méthode serait la même dans toutes les classes filles mais son implémentation diffèrerait. Pour imposer aux classes filles d'implémenter la méthode M :

- on va déclarer la signature de la méthode M dans la classe parent. Comme celle-ci ne sait pas comment l'implémenter, on préfixe la méthode par le mot clé **abstract** : cela signifie que l'implémentation de la méthode M est reportée sur les classes filles ;
- parce que la classe parent n'est pas totalement implémentée, elle est elle-aussi déclarée abstraite avec le même mot clé **abstract**. Ceci fait que la classe ne peut plus être instanciée. Il faut obligatoirement créer une classe fille qui définira l'implémentation de la méthode M, pour que le corps de la classe parent soit utilisable ;

Voici un exemple **[classes-15.php]** :

```php
<?php

// respect strict du type des paramètres des fonctions
declare(strict_types=1);

// classe principale abstraite
abstract Class Classe1 {

  // méthode connue de toutes les classes dérivées
  public function f(): int {
    return 1;
  }

  // méthode g abstraite - sera définie par les classes dérivées
  abstract function g(): int;
}

// classe dérivée
Class Classe2 extends Classe1 {

  // la méthode g de la classe parent doit être définie
  public function g(): int {
    return parent::f() + 10;
  }

}

// classe dérivée
Class Classe3 extends Classe1 {

  // la méthode g de la classe parent doit être définie
  public function g(): int {
    return parent::f() + 20;
  }

  // on peut redéfinir la méthode f de la classe parent
  public function f(): int {
    return 2;
  }

}

// code
$c2 = new Classe2();
print $c2->f() . "\n";
print $c2->g() . "\n";
$c3 = new Classe3();
print $c3->f() . "\n";
print $c3->g() . "\n";
```

Commentaires

- lignes 7-16 : la classe **[Classe1]** est abstraite (ligne 7) car elle ne sait pas implémenter la méthode g de la ligne 15. Elle doit donc être obligatoirement dérivée pour être utilisable ;
- lignes 19-26 : la classe **[Classe2]** étend la classe **[Classe1]** et redéfinit la méthode **g** de sa classe parent (lignes 22-24) ;
- lignes 29-41 : la classe **[Classe3]** étend la classe **[Classe1]** et redéfinit la méthode **g** de sa classe parent (lignes 32-34) ;
- lignes 37-39 : la classe **[Classe3]** redéfinit la méthode **f** de sa classe parent ;
- lignes 44-49 : on crée deux objets de type **[Classe2]** et **[Classe3]** et on appelle leurs méthodes **f** et **g** ;

Résultats

```
1   1
2   11
3   2
4   21
```

1.5.15 Classes finales

Une classe finale est une classe qu'on ne peut dériver. Considérons le script **[classes-11.php]** suivant :

```
1.  <?php
2.
3.  // espace de noms
4.
5.  namespace Exemples;
6.
7.  // classe non dérivable
8.  final class Classe1 {
9.
10. }
11.
12. // classe dérivée
13. class Classe2 extends Classe1 {
14.
15. }
16.
17. // code - doit provoquer une erreur
18. new Classe2();
```

Commentaires

- lignes 7-8 : le mot clé **final**, fait de la classe **[Classe1]** une classe finale qu'on ne peut dériver ;
- lignes 13-15 : la classe **[Classe2]** étend la classe finale **[Classe1]**, ce qui est une erreur ;
- ligne 8 : l'erreur ne sera signalée qu'à l'exécution du script lorsqu'on essaiera de manipuler un objet de type **[Classe2]** ;

Résultats

```
1.  Fatal error: Class Exemples\Classe2 may not inherit from final class (Exemples\Classe1) in C:\Data\st-
    2019\dev\php7\php5-exemples\exemples\classes\classes-11.php on line 14
```

1.5.16 Méthodes finales

Une méthode finale est une méthode qu'on ne peut redéfinir par dérivation. Voici un exemple **[classes-12.php]** :

```
1.  <?php
2.
3.  // respect strict du type des paramètres des fonctions
4.  declare(strict_types=1);
5.
6.  // espace de noms
7.  namespace Exemples;
8.
9.  // classe principale
10. class Classe1 {
11.
12.   // cette méthode ne peut être redéfinie dans une classe dérivée
13.   public final function f(): int {
14.     return 1;
15.   }
16.
17. }
18.
19. // classe dérivée
```

```
20. Class Classe2 extends Classe1 {
21.
22.   public function f(): int {
23.     return 2;
24.   }
25.
26. }
27.
28. // code - doit provoquer une erreur
29. new Classe2();
```

Commentaires

- ligne 13 : la méthode **f** de la classe **[Classe1]** est déclarée finale par le mot clé **final** ;
- ligne 20 : la classe **[Classe2]** étend classe **[Classe1]** ;
- lignes 22-23 : on redéfinit la fonction **f** de la classe parent **[Classe1]**. Cela doit provoquer une erreur ;
- ligne 29 : on crée un objet de type **[Classe2]** pour forcer l'interpréteur PHP à inspecter la classe **[Classe2]** ;

Résultats

```
1   Fatal error: Cannot override final method Exemples\Classe1::f() in C:\Data\st-2019\dev\php7\php5-
    exemples\exemples\classes\classes-12.php on line 26
```

1.5.17 Méthodes et attributs statiques

Une méthode **statique** est une méthode liée à la **classe** dans laquelle elle est définie et non aux objets instances de la classe. Ainsi si la classe **C** déclare une méthode statique **M**, pour utiliser cette dernière on écrira :

- **C::M** si on est à l'extérieur de la classe ;
- **self::M** si on est dans la classe ;

Voici un exemple **[classes-17.php]** :

```
1.  <?php
2.
3.  class Classe1 {
4.
5.    // méthode statique
6.    static function say(string $message): void {
7.      print "$message\n";
8.    }
9.
10. }
11.
12. // test -------------------
13. Classe1::say("hello");
```

Commentaires

- ligne 6 : la méthode **[say]** est déclarée statique avec le mot clé **static** ;
- ligne 13 : appel de la méthode statique **[say]** avec la notation : **Classe1::say** ;

Résultats

```
1   hello
```

Considérons maintenant le code suivant **[classes-18.php]** :

```
1.  <?php
2.
3.  class Classe1 {
4.    // attribut statique
5.    private static $nbObjects = 0;
6.
7.    public function __construct() {
8.      print "constructeur Classe1\n";
9.      self::$nbObjects++;
10.   }
11.
12.   // méthode statique
13.   static function say(): void {
```

```
14.     print self::$nbObjects ." objets de type [Classe1] ont été construits\n";
15.   }
16.
17. }
18.
19. // test -------------------
20. new Classe1();
21. new Classe1();
22. Classe1::say();
```

Commentaires

- ligne 5 : on déclare un attribut **statique** qui va compter le nombre d'instances de la classe **[Classe1]** créées. Ce n'est pas un attribut qui peut appartenir à une instance de la classe. En effet, si deux objets O1 et O2 sont créés, aucun des deux n'a connaissance de l'autre. Avoir un compteur dans l'instance n'a pas de sens : lorsqu'un nouvel objet est créé, dans quelle instance va-t-on incrémenter un compteur ? On serait obliger d'incrémenter le compteur d'un objet particulier délaissant les compteurs des autres instances. Un attribut **statique** est un attribut de classe et non d'instance de la classe ;
- lignes 7-10 : c'est dans le constructeur qu'on va compter les objets créés puisque la création de chaque nouvel objet provoque l'exécution du constructeur ;
- ligne 14 : on notera la notation **self::$nbObjects** pour indiquer qu'on fait référence à un attribut statique de la classe dans laquelle se trouve le code exécuté ;
- lignes 13-15 : la méthode statique **[say]** a pour rôle d'afficher le nombre d'objets créés ;
- lignes 20-22 : on crée deux objets et on fait afficher le compteur d'objets ;

Résultats

```
1   constructeur Classe1
2   constructeur Classe1
3   2 objets de type [Classe1] ont été construits
```

1.5.18 Visibilité entre classe Parent et classe Fille

Examinons le script **[classes-19.php]** suivant :

```
1.  <?php
2.
3.  class SomeParent {
4.    // attribut
5.    private $attributeOfParent = 4;
6.
7.    // méthode
8.    public function doTest(): void {
9.      // qui appelle ?
10.     print "parent :\n";
11.     var_dump($this);
12.     // affichage parent
13.     print "parent : attributeOfParent={$this->attributeOfParent}\n";
14.     print "parent : attributeOfChild={$this->attributeOfChild}\n";
15.   }
16.
17. }
18.
19. class SomeChild extends SomeParent {
20.   // attribut
21.   private $attributeOfChild = 14;
22.
23.   // méthode
24.   public function doTest(): void {
25.     // affichage enfant
26.     print "child : attributeOfParent={$this->attributeOfParent}\n";
27.     print "child : attributeOfChild={$this->attributeOfChild}\n";
28.
29.     // parent
30.     parent::doTest();
31.   }
32. }
33.
34. // script principal
35. print "--------------------------test1\n";
36. (new SomeParent())->doTest();
37. print "--------------------------test2\n";
38. (new SomeChild())->doTest();
```

Commentaires

- lignes 3-17 : la classe **[SomeParent]** ;
- lignes 19-32 : la classe fille **[SomeChild]**. On voit qu'elle étend la classe **[SomeParent]** (ligne 19) ;
- ligne 5 : la classe **[SomeParent]** n'a qu'un attribut ;
- lignes 8-17 : la méthode **[SomeParent::doTest]** a pour but d'afficher deux attributs :
  - **[$attributeOfParent]** qui appartient à la classe **[SomeParent]** ;
  - **[$attributeOfChild]** qui appartient à la classe **[SomeChild]** (ligne 21) ;
- lignes 10-11 : on affiche l'identité de l'appelant : on va en effet appeler la méthode de deux façons différentes :
  - à partir de la classe parent **[SomeParent]** ;
  - à partir de la classe fille **[SomeChild]** ;
- lignes 13-14 : affichage des deux attributs ;
- lignes 19-32 : la classe fille **[SomeChild]** qui étend la classe **[SomeParent]** (ligne 19) ;
- ligne 21 : la classe **[SomeChild]** n'a qu'un attribut ;
- lignes 24 : la méthode **[SomeChild::doTest]** a pour but d'afficher deux attributs :
  - **[$attributeOfParent]** qui appartient à la classe **[SomeParent]** ;
  - **[$attributeOfChild]** qui appartient à la classe **[SomeChild]** ;
- lignes 26-27 : affichage des deux attributs ;
- ligne 30 : appel de la méthode **[doTest]** de la classe parent qui va à son tour afficher les deux attributs ;
- ligne 36 : la méthode **[SomeParent::doTest]** est appelée ;
- ligne 38 : la méthode **[SomeChild::doTest]** est appelée ;

Dans le 1er test, la visibilité des deux attributs est **[private]**. On peut donc s'attendre à ce que la classe fille ne voit pas l'attribut de sa classe parent. Il faudrait que celui-ci ait au moins la visibilité **[protected]**. Mais qu'en est-il de l'attribut de la classe fille ? Est-il visible dans la classe parent ?

Voici les résultats de ce 1er test :

```
1   ---------------------------test1
2   parent :
3   object(SomeParent)#1 (1) {
4     ["attributeOfParent":"SomeParent":private]=>
5     int(4)
6   }
7   parent : attributeOfParent=4
8
9   Notice: Undefined property: SomeParent::$attributeOfChild in C:\Data\st-2019\dev\php7\poly\scripts-
    console\classes\classes-19.php on line 14
10  parent : attributeOfChild=
11  ---------------------------test2
12
13  Notice: Undefined property: SomeChild::$attributeOfParent in C:\Data\st-2019\dev\php7\poly\scripts-
    console\classes\classes-19.php on line 26
14  child : attributeOfParent=
15  child : attributeOfChild=14
16  parent :
17  object(SomeChild)#1 (2) {
18    ["attributeOfChild":"SomeChild":private]=>
19    int(14)
20    ["attributeOfParent":"SomeParent":private]=>
21    int(4)
22  }
23  parent : attributeOfParent=4
24
25  Fatal error: Uncaught Error: Cannot access private property SomeChild::$attributeOfChild in C:\Data\st-
    2019\dev\php7\poly\scripts-console\classes\classes-19.php:14
26  Stack trace:
27  #0 C:\Data\st-2019\dev\php7\poly\scripts-console\classes\classes-19.php(30): SomeParent->doTest()
28  #1 C:\Data\st-2019\dev\php7\poly\scripts-console\classes\classes-19.php(39): SomeChild->doTest()
29  #2 {main}
30    thrown in C:\Data\st-2019\dev\php7\poly\scripts-console\classes\classes-19.php on line 14
```

Commentaires

- lignes 1-10 : résultats du 1er test où la méthode **[SomeParent::doTest]** est appelée ;
- lignes 3-6 : on voit que l'objet qui appelle la méthode est de type **[SomeParent]** ;
- ligne 7 : affichage de l'attribut **[$attributeOfParent]** ;
- lignes 9-10 : on voit que l'attribut **[SomeParent::$attributeOfChild]** n'existe pas. Il n'est donc pas affiché ;
- lignes 11-30 : résultats du 2e test où la méthode **[SomeChild::doTest]** est appelée ;

- lignes 13-14 : on voit que l'attribut **[SomeChild::$attributeOfParent]** n'existe pas. Il n'est donc pas affiché. C'est normal : l'attribut **[SomeParent::$attributeOfParent]** est **[private]** et donc pas connu dans la classe fille ;
- ligne 15 : affichage de l'attribut **[$attributeOfChild]** ;
- lignes 16-30 : on est dans la méthode **[SomeParent::doTest]** appelée par la classe fille ;
- lignes 17-22 : on voit que **[$this]** est de type **[SomeChild]** avec deux attributs privés ;
- ligne 23 : de façon très étonnante, **[$this]** de type **[SomeChild]** voit ici l'attribut du parent **[$attributeOfParent]** ;
- lignes 25-30 : de façon tout aussi étonnante, **[$this]** de type **[SomeChild]** ne voit pas son attribut **[$attributeOfChild]** ;

Ce résultat est très étonnant : bien que les lignes 17-21 indiquent que **[$this]** est de type **[SomeChild]**, ce **[$this]** à l'intérieur de la méthode **[SomeParent::doTest]** se comporte comme s'il était une instance de la classe **[SomeParent]** et non de la classe **[SomeChild]**.

Faisons un nouveau test avec le script **[classes-20.php]**. L'attribut **[$attributeOfParent]** a maintenant une visibilité **[protected]** (ligne 6) :

```
1.  <?php
2.
3.  class SomeParent {
4.
5.    // attribut
6.    protected $attributeOfParent = 4;
7.
8.    // méthode
9.    public function doTest(): void {
10.     // qui appelle ?
11.     print "parent :\n";
12.     var_dump($this);
13.     // affichage parent
14.     print "parent : attributeOfParent={$this->attributeOfParent}\n";
15.     print "parent : attributeOfChild={$this->attributeOfChild}\n";
16.   }
17.
18. }
19.
20. class SomeChild extends SomeParent {
21.
22.   // attribut
23.   private $attributeOfChild = 14;
24.
25.   // méthode
26.   public function doTest(): void {
27.     // affichage enfant
28.     print "child : attributeOfParent={$this->attributeOfParent}\n";
29.     print "child : attributeOfChild={$this->attributeOfChild}\n";
30.
31.     // parent
32.     parent::doTest();
33.   }
34.
35. }
36.
37. // script principal
38. print "--------------------------test1\n";
39. (new SomeParent())->doTest();
40. print "--------------------------test2\n";
41. (new SomeChild())->doTest();
```

Résultats

```
1   --------------------------test1
2   parent :
3   object(SomeParent)#1 (1) {
4     ["attributeOfParent":protected]=>
5     int(4)
6   }
7   parent : attributeOfParent=4
8
9   Notice: Undefined property: SomeParent::$attributeOfChild in C:\Data\st-2019\dev\php7\poly\scripts-
    console\classes\classes-20.php on line 14
10  parent : attributeOfChild=
11  --------------------------test2
12  child : attributeOfParent=4
13  child : attributeOfChild=14
14  parent :
15  object(SomeChild)#1 (2) {
```

```
16    ["attributeOfChild":"SomeChild":private]=>
17    int(14)
18    ["attributeOfParent":protected]=>
19    int(4)
20  }
21  parent : attributeOfParent=4
22
23  Fatal error: Uncaught Error: Cannot access private property SomeChild::$attributeOfChild in C:\Data\st-
    2019\dev\php7\poly\scripts-console\classes\classes-20.php:14
24  Stack trace:
25  #0 C:\Data\st-2019\dev\php7\poly\scripts-console\classes\classes-20.php(30): SomeParent->doTest()
26  #1 C:\Data\st-2019\dev\php7\poly\scripts-console\classes\classes-20.php(39): SomeChild->doTest()
27  #2 {main}
28    thrown in C:\Data\st-2019\dev\php7\poly\scripts-console\classes\classes-20.php on line 14
```

Commentaires

- ligne 12 : la classe **[SomeChild]** voit désormais l'attribut de son parent **[$attributeOfParent]**. C'est normal puisque celui-ci a maintenant une portée **[protected]** ;
- dans la méthode **[someParent::doTest]**, l'objet **[$this]** est de type **[SomeChild]** (lignes 15-20). Il voit l'attribut de son parent **[$attributeOfparent]** (ligne 21) mais toujours pas son propre attribut **[$attributeOfChild]** (lignes 23-28) ;

Dans le 3e essai **[classes-21.php]**, l'attribut **[$attributeOfChild]** a lui également une portée **[protected]** :

```
1.  <?php
2.
3.  class SomeParent {
4.    // attribut
5.    protected $attributeOfParent = 4;
6.
7.    // méthode
8.    public function doTest(): void {
9.      // qui appelle ?
10.     print "parent :\n";
11.     var_dump($this);
12.     // affichage parent
13.     print "parent : attributeOfParent={$this->attributeOfParent}\n";
14.     print "parent : attributeOfChild={$this->attributeOfChild}\n";
15.   }
16.
17. }
18.
19. class SomeChild extends SomeParent {
20.   // attribut
21.   protected $attributeOfChild = 14;
22.
23.   // méthode
24.   public function doTest(): void {
25.     // affichage enfant
26.     print "child : attributeOfParent={$this->attributeOfParent}\n";
27.     print "child : attributeOfChild={$this->attributeOfChild}\n";
28.
29.     // parent
30.     parent::doTest();
31.   }
32.
33. }
34.
35. // script principal
36. print "--------------------------test1\n";
37. (new SomeParent())->doTest();
38. print "--------------------------test2\n";
39. (new SomeChild())->doTest();
```

Les résultats de l'exécution sont les suivants :

```
1   --------------------------test1
2   parent :
3   object(SomeParent)#1 (1) {
4     ["attributeOfParent":protected]=>
5     int(4)
6   }
7   parent : attributeOfParent=4
8
```

```
9   Notice: Undefined property: SomeParent::$attributeOfChild in C:\Data\st-2019\dev\php7\poly\scripts-
    console\classes\classes-21.php on line 14
10  parent : attributeOfChild=
11  --------------------------test2
12  child : attributeOfParent=4
13  child : attributeOfChild=14
14  parent :
15  object(SomeChild)#1 (2) {
16    ["attributeOfChild":protected]=>
17    int(14)
18    ["attributeOfParent":protected]=>
19    int(4)
20  }
21  parent : attributeOfParent=4
22  parent : attributeOfChild=14
```

- ligne 22 : cette fois-ci, à l'intérieur du parent, **[$this]** de type **[SomeChild]** (lignes 15-20) voit l'attribut protégé **[$attributeOfChild]** de sa propre classe **[SomeChild]**.

Que retenir de ces tests ?

- que le **[$this]** instance d'une classe parent, utilisée dans une méthode de la classe parent voit :
  - les attributs et méthodes de la classe parent quelque soit leur visibilité ;
  - ne voit rien des attributs et méthodes de ses classes filles ;

  C'est le comportement attendu.

- que le **[$this]** instance d'une classe fille, utilisée dans une méthode de la classe fille voit :
  - les attributs et méthodes de la classe parent s'ils ont au moins la visibilité **[protected]**. Ceux qui ont la visibilité **[private]** ne sont pas vus ;
  - les attributs et méthodes de la classe fille quelque soit leur visibilité ;

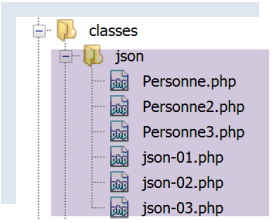
  C'est le comportement attendu.

- que le **[$this]** instance d'une classe fille, utilisée dans une méthode de la classe parent voit :
  - les attributs et méthodes de la classe parent quelque soit leur visibilité ;
  - les attributs et méthodes de sa propre classe uniquement s'ils ont au moins la visibilité **[protected]**. Ceux qui ont la visibilité **[private]** ne sont pas vus ;

  C'est un comportement inattendu.

1.5.19 Encodage jSON d'une classe

Dans une classe la méthode **[__toString]** est souvent présente : elle est censée rendre une chaîne de caractères représentant l'objet qui l'appelle. Il peut être tentant que cette chaîne soit une chaîne jSON. Nous explorons cette voie maintenant.



Nous utiliserons la classe **[Personne]** suivante :

```
1.  <?php
2.
3.  class Personne {
4.    // attributs
5.    private $nom;
6.    private $prénom;
7.    private $âge;
8.    private $enfants;
9.
10.   // setter global
11.   public function setFromArray(array $arrayOfAttributes): Personne {
12.     // initialisation de certains attributs de la classe
```

```
13.     foreach ($arrayOfAttributes as $attribute => $value) {
14.       $this->$attribute = $value;
15.     }
16.     // on retourne l'objet
17.     return $this;
18.   }
19.
20.   // getters
21.   public function getNom() {
22.     return $this->nom;
23.   }
24.
25.   public function getPrénom() {
26.     return $this->prénom;
27.   }
28.
29.   public function getÂge() {
30.     return $this->âge;
31.   }
32.
33.   public function getEnfants() {
34.     return $this->enfants;
35.   }
36.
37.   // __toString
38.   public function __toString(): string {
39.     // on identifie l'objet
40.     var_dump($this);
41.     // on récupère ses attributs
42.     $attributes = \get_object_vars($this);
43.     var_dump($attributes);
44.     // on rend la chaîne JSON des attributs
45.     return \json_encode($attributes, JSON_UNESCAPED_UNICODE);
46.   }
47.
48. }
```

Commentaires

- lignes 5-8 : les quatre attributs de la classe ;
- lignes 20-35 : les getters qui permettent d'avoir la valeur de ces attributs ;
- lignes 11-18 : un setter global qui permet d'initialiser les attributs à partir d'un tableau associatif **`[$arrayOfAttributes]`** ayant des clés **identiques** aux attributs de la classe ;
- lignes 38-46 : la méthode **`[__toString]`** de la classe ;
- ligne 42 : la fonction PHP **`[get_object_vars]`** permet d'obtenir la valeur des attributs de la classe sous la forme d'un tableau associatif **`['nom'=>'nom1', 'prénom'=>'prénom1, 'âge'=>âge1, 'enfants'=>[]]`** ;
- ligne 45 : on rend la chaîne jSON de ce tableau d'attributs ;

Examinons le script **`[json-01.php]`** qui exploite la classe **`[Personne]`** :

```
1.  <?php
2.
3.  // classe Personne
4.  require "Personne.php";
5.
6.  // instanciation du père
7.  $père = new Personne();
8.  // initialisation du père
9.  $père->setFromArray([
10.   "nom" => "Bertholomé",
11.   "prénom" => "Dieudonné",
12.   "âge" => 58
13. ]);
14. // instanciation et initialisation enfant1
15. $enfant1 = (new Personne())->setFromArray([
16.   "nom" => "Bertholomé",
17.   "prénom" => "Sylvain",
18.   "âge" => 17
19.   ]);
20. // instanciation et initialisation enfant2
21. $enfant2 = (new Personne())->setFromArray([
22.   "nom" => "Bertholomé",
23.   "prénom" => "Géraldine",
24.   "âge" => 12
```

```
25.   ]);
26. // initialisation enfants du père
27. $père->setFromArray([
28.   "enfants" => [$enfant1, $enfant2]
29. ]);
30.
31. // affichage éléments du père
32. $enfant1=($père->getEnfants())[0];
33. $enfant2=($père->getEnfants())[1];
34. print "-----------------------enfant1\n";
35. print "enfant1=$enfant1\n";
36. print "-----------------------enfant2\n";
37. print "enfant2=$enfant2\n";
38. print "-----------------------père\n";
39. print "père=$père\n";
```

Commentaires

- lignes 6-13 : on initialise un objet **[Personne] [$père]** avec la méthode **[Personne::setFromArray]** qui permet d'initialiser un objet **[Personne]** avec un tableau ayant des clés identiques aux attributs de la classe **[Personne]** ;
- lignes 14-19 : on initialise un objet **[Personne] [$enfant1]** de la même façon ;
- lignes 21-25 : on initialise un objet **[Personne] [$enfant2]** ;
- lignes 27-29 : on initialise l'attribut **[$père→enfants]** avec un tableau des deux enfants ;
- lignes 32-33 : on affecte les deux enfants au père ;
- ligne 35 : l'opération **[print]** va chercher à transformer l'objet **[$enfant1]** en chaîne de caractères. Pour cela, elle utilise la méthode **[__toString]** de l'objet. On espère donc voir la chaîne jSON de l'objet ;
- lignes 38-39 : on fait de même avec le père ;

Les résultats sont les suivants :

```
1   -----------------------enfant1
2   object(Personne)#2 (4) {
3     ["nom":"Personne":private]=>
4     string(11) "Bertholomé"
5     ["prénom":"Personne":private]=>
6     string(7) "Sylvain"
7     ["âge":"Personne":private]=>
8     int(17)
9     ["enfants":"Personne":private]=>
10    NULL
11  }
12  array(4) {
13    ["nom"]=>
14    string(11) "Bertholomé"
15    ["prénom"]=>
16    string(7) "Sylvain"
17    ["âge"]=>
18    int(17)
19    ["enfants"]=>
20    NULL
21  }
22  enfant1={"nom":"Bertholomé","prénom":"Sylvain","âge":17,"enfants":null}
23  -----------------------enfant2
24  object(Personne)#3 (4) {
25    ["nom":"Personne":private]=>
26    string(11) "Bertholomé"
27    ["prénom":"Personne":private]=>
28    string(10) "Géraldine"
29    ["âge":"Personne":private]=>
30    int(12)
31    ["enfants":"Personne":private]=>
32    NULL
33  }
34  array(4) {
35    ["nom"]=>
36    string(11) "Bertholomé"
37    ["prénom"]=>
38    string(10) "Géraldine"
39    ["âge"]=>
40    int(12)
41    ["enfants"]=>
42    NULL
43  }
44  enfant2={"nom":"Bertholomé","prénom":"Géraldine","âge":12,"enfants":null}
```

```
45  ------------------------père
46  object(Personne)#1 (4) {
47    ["nom":"Personne":private]=>
48    string(11) "Bertholomé"
49    ["prénom":"Personne":private]=>
50    string(10) "Dieudonné"
51    ["âge":"Personne":private]=>
52    int(58)
53    ["enfants":"Personne":private]=>
54    array(2) {
55      [0]=>
56      object(Personne)#2 (4) {
57        ["nom":"Personne":private]=>
58        string(11) "Bertholomé"
59        ["prénom":"Personne":private]=>
60        string(7) "Sylvain"
61        ["âge":"Personne":private]=>
62        int(17)
63        ["enfants":"Personne":private]=>
64        NULL
65      }
66      [1]=>
67      object(Personne)#3 (4) {
68        ["nom":"Personne":private]=>
69        string(11) "Bertholomé"
70        ["prénom":"Personne":private]=>
71        string(10) "Géraldine"
72        ["âge":"Personne":private]=>
73        int(12)
74        ["enfants":"Personne":private]=>
75        NULL
76      }
77    }
78  }
79  array(4) {
80    ["nom"]=>
81    string(11) "Bertholomé"
82    ["prénom"]=>
83    string(10) "Dieudonné"
84    ["âge"]=>
85    int(58)
86    ["enfants"]=>
87    array(2) {
88      [0]=>
89      object(Personne)#2 (4) {
90        ["nom":"Personne":private]=>
91        string(11) "Bertholomé"
92        ["prénom":"Personne":private]=>
93        string(7) "Sylvain"
94        ["âge":"Personne":private]=>
95        int(17)
96        ["enfants":"Personne":private]=>
97        NULL
98      }
99      [1]=>
100     object(Personne)#3 (4) {
101       ["nom":"Personne":private]=>
102       string(11) "Bertholomé"
103       ["prénom":"Personne":private]=>
104       string(10) "Géraldine"
105       ["âge":"Personne":private]=>
106       int(12)
107       ["enfants":"Personne":private]=>
108       NULL
109     }
110   }
111 }
112 père={"nom":"Bertholomé","prénom":"Dieudonné","âge":58,"enfants":[{},{}]}
```

Commentaires

- lignes 2-11 : l'objet **`[$enfant1]`** ;
- lignes 12-21 : le tableau des attributs de l'objet **`[$enfant1]`**. On les a bien tous ;
- ligne 22 : on a bien la chaîne jSON de l'objet **`[$enfant1]`** ;
- lignes 23-44 : idem pour l'objet **`[$enfant2]`** ;

- lignes 45-112 : pour le père c'est un peu différent puisque son attribut **[enfants]** n'est pas NULL comme il l'était chez les enfants ;
- ligne 112 : on voit que dans la chaîne jSON du père, il manque les enfants ;
- lignes 79-111 : on voit que dans le tableau d'attributs du père, l'enfant 1 (lignes 89-98) est resté un objet, de même pour l'enfant 2 (lignes 99-110). En clair, l'expression **[\get_object_vars($this)]** où **[$this]** représente le père n'est pas récursive : si un attribut de la classe **[Personne]** est lui-même un objet, l'expression **[\get_object_vars($this)]** n'essaie pas d'obtenir son tableau d'attributs ;

On peut améliorer les choses. On modifie la classe **[Personne]** en la classe **[Personne2]** suivante :

```
1.  <?php
2.
3.  class Personne2 {
4.    // attributs
5.    private $nom;
6.    private $prénom;
7.    private $âge;
8.    private $enfants;
9.
10.   // setter global
11.   public function setFromArray(array $arrayOfAttributes): Personne2 {
12.     // initialisation de certains attributs de la classe
13.     foreach ($arrayOfAttributes as $attribute => $value) {
14.       $this->$attribute = $value;
15.     }
16.     // on retourne l'objet
17.     return $this;
18.   }
19.
20.   // getters
21.   public function getNom() {
22.     return $this->nom;
23.   }
24.
25.   public function getPrénom() {
26.     return $this->prénom;
27.   }
28.
29.   public function getÂge() {
30.     return $this->âge;
31.   }
32.
33.   public function getEnfants() {
34.     return $this->enfants;
35.   }
36.
37.   // __toString
38.   public function __toString(): string {
39.     // on récupère les attributs de l'objet
40.     $attributes = $this->getAttributes($this);
41.     $enfants = $attributes["enfants"];
42.     if ($enfants != NULL) {
43.       $attributes["enfants"] = [$enfants[0]->getAttributes(), $enfants[1]->getAttributes()];
44.     }
45.     // on rend la chaîne JSON des attributs
46.     return \json_encode($attributes, JSON_UNESCAPED_UNICODE);
47.   }
48.
49.   public function getAttributes(): array {
50.     return \get_object_vars($this);
51.   }
52.
53. }
```

Commentaires

- lignes 49-51 : la fonction **[getAttributes]** rend le tableau des attributs de l'objet qui l'appelle ;
- lignes 38-47 : la fonction **[__toString]** ;
- ligne 40 : on récupère les attributs de la classe **[Personne]** dans le tableau **[$attributes]** ;
- ligne 41 : on sait, d'après l'exemple précédent, que **[$attributes["enfants"]]** est un tableau de deux objets de type **[Personne]** ;
- lignes 42-44 : les deux objets sont remplacés par leur tableau d'attributs ;
- ligne 46 : il ne reste plus qu'à encoder en jSON le tableau des attributs construit ;

Le script **[json-02.php]** exploite la classe **[Personne2]** de la façon suivante :

```
1.  <?php
2.
3.  // classe Personne2
4.  require "Personne2.php";
5.
6.  // instanciation du père
7.  $père = new Personne2();
8.  // initialisation
9.  $père->setFromArray([
10.   "nom" => "Bertholomé",
11.   "prénom" => "Dieudonné",
12.   "âge" => 58
13. ]);
14. // instanciation et initialisation enfant1
15. $enfant1 = (new Personne2())->setFromArray([
16.   "nom" => "Bertholomé",
17.   "prénom" => "Sylvain",
18.   "âge" => 17
19.   ]);
20. // instanciation et initialisation enfant2
21. $enfant2 = (new Personne2())->setFromArray([
22.   "nom" => "Bertholomé",
23.   "prénom" => "Géraldine",
24.   "âge" => 12
25.   ]);
26. // initialisation enfants du père
27. $père->setFromArray([
28.   "enfants" => [$enfant1, $enfant2]
29. ]);
30.
31. // affichage père
32. print "------------------------père\n";
33. print "père=$père\n";
```

Le script **[json-02.php]** est identique au script **[json-01.php]** si ce n'est que la classe **[Personne2]** a remplacé la classe **[Personne]**.

Les résultats de l'exécution sont les suivants :

```
1   ------------------------père
2   père={"nom":"Bertholomé","prénom":"Dieudonné","âge":58,"enfants":[{"nom":"Bertholomé","prénom":"Sylvain","â
    ge":17,"enfants":null},{"nom":"Bertholomé","prénom":"Géraldine","âge":12,"enfants":null}]}
```

Cette fois, on a bien obtenu les enfants avec le père.

La solution précédente n'est pas satisfaisante car les enfants peuvent avoir eux-mêmes des enfants. On retrouve alors le problème précédent.

La classe **[Personne3]** résoud ce problème de la façon suivante :

```
1.  <?php
2.
3.  class Personne3 {
4.    // attributs
5.    private $nom;
6.    private $prénom;
7.    private $âge;
8.    private $enfants;
9.
10.   // setter global
11.   public function setFromArray(array $arrayOfAttributes): Personne3 {
12.     …
13.   }
14.
15.   // getters
16.   …
17.
18.   // __toString
19.   public function __toString(): string {
20.     // on rend la chaîne JSON des attributs
21.     $attributes = [];
22.     $this->getRecursiveAttributes($attributes, $this, []);
```

```
23.     // chaîne JSON des attributs
24.     return \json_encode($attributes, JSON_UNESCAPED_UNICODE);
25.   }
26.
27.   public function getAttributes(): array {
28.     return \get_object_vars($this);
29.   }
30.
31.   private function getRecursiveAttributes(array &$attributes, $value, $keys): void {
32.     // analyse de la valeur [$value]
33.     // $keys est un tableau [key1, key2, .., keyn]
34.     // $value=$attributes[key1][key2]....[keyn]
35.     // si [$value] est un objet on utilise sa méthode [getAttributes]
36.     if (\is_object($value)) {
37.       // attributs de l'objet [$value]
38.       $objectAttributes = $value->getAttributes();
39.       // y-a-t-il des clés ?
40.       if ($keys) {
41.         // dans [$attributes], on va remplacer $value par le tableau de ses attributs
42.         // il faut construire l'élément $attributes[key1][key2]...[keyn]
43.         // où $keys est le tableau [key1, key2, .., keyn]
44.         // on prend la référence du tableau [$attributes]
45.         $attribute = &$attributes;
46.         // on scanne le tableau des clés
47.         foreach ($keys as $key) {
48.           // on prend la référence de l'attribut
49.           $attribute = &$attribute[$key];
50.         }
51.         // ici $attribut et $attributes[key1][key2]...[key(n)] sont identiques
52.         // ils partagent le même emplacement mémoire
53.         // l'objet [$value] est remplacé par son tableau d'attributs;
54.         // il faut écrire $attributes[key1][key2]...[keyn]=$objectAttributes
55.         // ce qui équivaut à $attribute = $objectAttributes
56.         $attribute = $objectAttributes;
57.       } else {
58.         // pas de clés - on est au début de l'exploration de l'objet
59.         // $objectAttributes représente les attributs de 1er niveau de la classe
60.         $attributes += $objectAttributes;
61.       }
62.       // peut-être que dans [$objectAttributes] il y a encore des objets
63.       // on explore les attributs de [$objectAttributes]
64.       $this->getRecursiveAttributes($attributes, $objectAttributes, $keys);
65.     } else {
66.       if (\is_array($value)) {
67.         // on a un tableau - on analyse chacun de ses éléments
68.         foreach ($value as $key => $élément) {
69.           // on rajoute la clé courante au tableau $keys
70.           \array_push($keys, $key);
71.           // on analyse $élément
72.           $this->getRecursiveAttributes($attributes, $élément, $keys);
73.           // on enlève du tableau $keys la clé qui vient d'être analysée
74.           \array_pop($keys);
75.         }
76.       }
77.     }
78.   }
79.
80. }
```

Commentaires

- lignes 21-22 : cette fois-ci, la méthode **[__toString]** demande les attributs de sa classe et demande à ce que ce soit fait récursivement : si un attribut est un objet ou un tableau d'objets, alors chaque objet doit être remplacé par son tableau d'attributs dans le tableau d'attributs final de la classe ;
- lignes 31-78 : la fonction **[getRecursiveAttributes]** fait ce travail. On a commenté son code. L'écriture d'une fonction récursive est souvent quelque chose de complexe. C'est le cas ici. Le lecteur ne perdra rien s'il ne le comprend pas. Il existe des bibliothèques qui font ce travail. L'appel récursif a lieu aux lignes 64 et 72 ;
- l'intérêt de ce code est qu'il n'a pas été écrit pour la seule classe **[Personne3]**. Il est valable pour toute classe ayant des attributs ayant pour valeurs des types d'objets différents, tant que les classes utilisées par la classe principale, ont comme elle la méthode **[getAttributes]** des lignes 27-29

Le script **[json-03.php]** utilise la classe **[Personne3]** de la façon suivante :

```
1.  <?php
```

```
2.
3.  // classe Personne3
4.  require "Personne3.php";
5.
6.  // instanciation du père
7.  $père = new Personne3();
8.  // initialisation
9.  $père->setFromArray([
10.   "nom" => "Bertholomé",
11.   "prénom" => "Dieudonné",
12.   "âge" => 58
13. ]);
14. // instanciation et initialisation enfant1
15. $enfant1 = (new Personne3())->setFromArray([
16.   "nom" => "Bertholomé",
17.   "prénom" => "Sylvain",
18.   "âge" => 27
19.   ]);
20. // instanciation et initialisation enfant2
21. $enfant2 = (new Personne3())->setFromArray([
22.   "nom" => "Bertholomé",
23.   "prénom" => "Géraldine",
24.   "âge" => 12
25.   ]);
26. // initialisation enfants du père
27. $père->setFromArray([
28.   "enfants" => [$enfant1, $enfant2]
29. ]);
30. // instanciation et initialisation enfant11
31. $enfant11 = (new Personne3())->setFromArray([
32.   "nom" => "Bertholomé",
33.   "prénom" => "Gaëtan",
34.   "âge" => 2
35.   ]);
36. // instanciation et initialisation enfant2
37. $enfant12 = (new Personne3())->setFromArray([
38.   "nom" => "Bertholomé",
39.   "prénom" => "Mathilde",
40.   "âge" => 1
41.   ]);
42. // initialisation enfants de enfant1
43. $enfant1->setFromArray([
44.   "enfants" => [$enfant11, $enfant12]
45. ]);
46. // affichage père
47. print "------------------------père\n";
48. print "père=$père\n";
```

- lignes 30-45 : on donne deux enfants à **[$enfant1]** ;

Les résultats de l'exécution sont les suivants :

```
1   ------------------------père
2   père={"nom":"Bertholomé","prénom":"Dieudonné","âge":58,"enfants":[{"nom":"Bertholomé","prénom":"Sylvain","â
    ge":27,"enfants":[{"nom":"Bertholomé","prénom":"Gaëtan","âge":2,"enfants":null},{"nom":"Bertholomé","prénom
    ":"Mathilde","âge":1,"enfants":null}]},{"nom":"Bertholomé","prénom":"Géraldine","âge":12,"enfants":null}]}
```

Si on met en forme ce résultat, on obtient la chose suivante :

```
1   père={
2           "nom": "Bertholomé",
3           "prénom": "Dieudonné",
4           "âge": 58,
5           "enfants": [
6                   {
7                           "nom": "Bertholomé",
8                           "prénom": "Sylvain",
9                           "âge": 27,
10                          "enfants": [
11                                  {
12                                          "nom": "Bertholomé",
13                                          "prénom": "Gaëtan",
14                                          "âge": 2,
15                                          "enfants": null
16                                  },
17                                  {
```

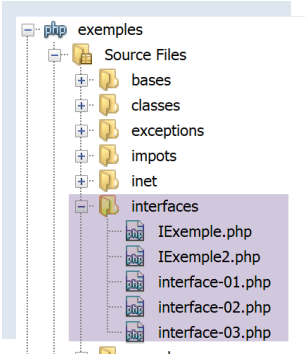
```
18                                      "nom": "Bertholomé",
19                                      "prénom": "Mathilde",
20                                      "âge": 1,
21                                      "enfants": null
22                                  }
23                          ]
24                  },
25                  {
26                          "nom": "Bertholomé",
27                          "prénom": "Géraldine",
28                          "âge": 12,
29                          "enfants": null
30                  }
31          ]
32  }
```

On a bien récupéré les chaîne jSON de tous les objets **[Personne]** formant le père.

1.6 Les interfaces

Une interface est une structure définissant des prototypes de méthodes. Une classe implémentant une interface doit définir le code de toutes les méthodes de l'interface.

1.6.1 L'arborescence des scripts



1.6.2 Une première interface

Nous définissons une interface **[IExemple]** dans un script **[IExemple.php]**. Dans ce même script, nous définissons deux classes implémentant l'interface **[IExemple]** :

```
1.  <?php
2.
3.  // respect strict des types des paramètres des fonctions
4.  declare(strict_types=1);
5.
6.  // espace de noms
7.  namespace Exemples;
8.
9.  // une interface
10. interface IExemple {
11.     // les méthodes de l'interface
12.     public function ajouter(int $i, int $j): int;
13.     public function soustraire(int $i, int $j): int;
14. }
15.
16. // implémentation 1 de l'interface IExemple
17. class Classe1 implements IExemple {
18.     public function ajouter(int $a, int $b): int {
19.         return $a + $b + 10;
20.     }
21.
22.     public function soustraire(int $a, int $b): int {
23.         return $a - $b + 20;
24.     }
25.
26. }
27.
```

```
28. // implémentation 2 de l'interface IExemple
29. class Classe2 implements IExemple {
30.     public function ajouter(int $a, int $b) : int{
31.         return $a + $b + 100;
32.     }
33.
34.     public function soustraire(int $a, int $b) : int {
35.         return $a - $b + 200;
36.     }
37.
38. }
```

Commentaires

- lignes 10-14 : l'interface *IExemple* définit deux méthodes : la fonction *ajouter* (ligne 12) et la fonction *soustraire* (ligne 13). L'interface ne définit pas le code de ces méthodes. Ce sont les classes implémentant l'interface qui vont le faire ;
- ligne 17 : la classe *Classe1* implémente (**implements**) l'interface *IExemple*. Elle définit donc un code pour les méthodes *ajouter* (ligne 18) et *soustraire* (ligne 22) ;
- lignes 29-38 : idem pour la classe *Classe2 ;*

Nous utilisons l'interface **[IExemple]** dans le script **[interface-01.php]** suivant :

```
1.  <?php
2.
3.  // respect strict des types des paramètres des fonctions
4.  declare(strict_types=1);
5.
6.  // inclusion et qualifications des classes et interfaces nécessaires au script
7.  require_once __DIR__."/IExemple.php";
8.  use \Exemples\Classe1;
9.  use \Exemples\Classe2;
10. use \Exemples\IExemple;
11.
12. // fonction
13. function calculer(int $a, int $b, IExemple $interface): void {
14.   print $interface->ajouter($a, $b) . "\n";
15.   print $interface->soustraire($a, $b) . "\n";
16. }
17.
18. // ------------------- main
19. // création de 2 objets de type Classe1 et Classe2
20. $c1 = new Classe1();
21. $c2 = new Classe2();
22. // appel de la fonction calculer
23. calculer(4, 3, $c1);
24. calculer(14, 13, $c2);
```

Commentaires

- lignes 7-10 : inclusion et définition des classes et interfaces utilisées par le script ;
- ligne 7 : au lieu de l'instruction **include** nous avons utilisé ici l'instruction **require_once**. L'instruction **include** inclut un fichier dans le script en cours même si celui-ci a déjà été inclus alors que l'instruction **require_once** ne l'inclut qu'une fois. Donc pour ce qui est de l'inclusion de code, on préfèrera l'instruction **require_once** ;
- lignes 13-16 : on définit une fonction ;
- ligne 13 : nous avons déclaré ici le type des trois paramètres. Si à l'exécution, le paramètre effectif n'est pas du type attendu, une erreur est lancée. Le 3e paramètre est de type **[IExemple]**. Cela veut dire que le paramètre effectif doit être une instance de classe implémentant l'interface **[IExemple]**, donc dans notre exemple, une instance des classes **[Classe1]** ou **[Classe2]** ;
- lignes 14-15 : nous utilisons les méthodes **[ajouter, soustraire]** de l'interface **[IExemple]** ;
- ligne 23 : le 3e paramètre effectif est de type **[Classe1]** compatible avec le type **[IExemple]** du 3e paramètre formel de la fonction ;
- ligne 24 : idem pour la classe **[Classe2]** ;

Résultats

```
1   17
2   21
3   127
4   201
```

1.6.3 Dérivation d'une interface

De la même façon qu'une classe fille peut étendre une classe parent, une interface fille peut étendre une interface parent.

Considérons l'interface **[IExemple2]** définie dans le fichier **[IExemple2.php]** :

```
1.  <?php
2.
3.  // respect strict des types des paramètres des fonctions
4.  declare(strict_types=1);
5.  // espace de noms
6.  namespace Exemples;
7.
8.  // inclusion de la définition de l'interface IExemple
9.  require_once __DIR__."/IExemple.php";
10.
11. // une interface qui dérive de IExemple
12. interface IExemple2 extends IExemple {
13.     // la méthode de l'interface
14.     public function multiplier(int $i, int $j): int;
15. }
16.
17. // implémentation de l'interface IExemple2
18. class Classe3 extends Classe2 implements IExemple2 {
19.
20.     public function multiplier(int $a, int $b): int {
21.         return $a * $b;
22.     }
23.
24. }
```

Commentaires

- ligne 6 : l'interface **[IExemple2]** sera dans le même espace de noms que l'interface **[Iexemple]** ;
- ligne 9 : on inclut le fichier de définition de l'interface **[Iexemple]** ;
- lignes 12-15 : définissent l'interface **[IExemple2]** qui étend (hérite) l'interface **[Iexemple]** (**extends**) et lui ajoute une nouvelle méthode (ligne 14);
- lignes 18-24 : la classe **[Classe3]** étend la classe **[Classe2]**. Comme celle-ci implémente l'interface **[IExemple]**, ce sera également le cas de la classe **[Classe3]**. En effet, **[Classe3]** hérite de toutes les méthodes publiques et protégées de la classe **[Classe2]**, donc des méthodes **ajouter** et **soustraire**. Par ailleurs, on indique que **[Classe3]** implémente l'interface **[IExemple2]**. A ce titre, elle doit également implémenter la méthode **multiplier**, ce que ne fait pas **[Classe2]**. C'est ce que font les lignes 20-22;

L'interface **[IExemple2]** est exploitée par le script **[interface-02]** suivant:

```
1.  <?php
2.
3.  // respect strict des types des paramètres des fonctions
4.  declare(strict_types=1);
5.
6.  // inclusion et qualifications des classes et interfaces nécessaires au script
7.  require_once __DIR__."/IExemple2.php";
8.  use \Exemples\IExemple2;
9.  use \Exemples\Classe3;
10.
11. // fonction
12. function calculer(int $a, int $b, IExemple2 $interface) {
13.   print $interface->ajouter($a, $b) . "\n";
14.   print $interface->soustraire($a, $b) . "\n";
15.   print $interface->multiplier($a, $b) . "\n";
16. }
17.
18. // ------------------- main
19. // création d'1 objet de type Classe3 qui implémente iExemple2
20. $c3 = new Classe3();
21. // appel de la fonction calculer
22. calculer(4, 3, $c3);
```

Commentaires

- ligne 12 : le 3e paramètre de la fonction **[calculer]** est de type **[IExemple2]** ;
- lignes 13-15 : on utilise les trois méthodes de l'interface **[IExemple2]** ;

- ligne 22 : on appelle la fonction **[calculer]** avec comme 3e paramètre un objet de type **[IExemple2]** (ligne 20) ;

Résultats

```
1   107
2   201
3   12
```

1.6.4 Passage d'une interface en paramètre d'une fonction

Nous avons vu au paragraphe que lorsque le type attendu pour un paramètre de fonction est une classe alors le paramètre effectif peut être un objet du type attendu **ou dérivé**. Il en est de même pour les interfaces comme le montre le script suivant **[interface-03.php]** :

```
1.  <?php
2.
3.  // respect strict des types des paramètres des fonctions
4.  declare(strict_types=1);
5.
6.  // inclusion et qualifications des classes et interfaces nécessaires au script
7.  require_once __DIR__."/IExemple.php";
8.  use \Exemples\IExemple;
9.  require_once __DIR__."/IExemple2.php";
10. use \Exemples\Classe3;
11.
12. // fonction
13. function calculer(int $a, int $b, IExemple $interface) {
14.   print $interface->ajouter($a, $b) . "\n";
15.   print $interface->soustraire($a, $b) . "\n";
16. }
17.
18. // ------------------- main
19. // création d'1 objet de type Classe3 qui implémente IExemple2 et donc IExemple
20. $c3 = new Classe3();
21. // appel de la fonction calculer
22. calculer(4, 3, $c3);
```

Commentaires

- ligne 13, la fonction **[calculer]** attend comme troisième paramètre un type **[IExemple]**. Cela signifie que le type du paramètre effectif pourra être de type **[IExemple] ou dérivé** ;
- ligne 20 : on instancie un objet $**c3** de type **[Classe3]**, classe qui implémente l'interface **[IExemple2]** qui elle-même étend l'interface **[IExemple]**. Donc finalement $**c3** implémente l'interface **[IExemple]** ;
- ligne 22 : on appelle la fonction **[calculer]** avec comme troisième paramètre un objet $**c3** de type **[Classe3]**. Par le jeu des héritages de classes et des implémentations, on a vu que l'objet $**c3** implémentait le type **[IExemple]**. On peut donc l'utiliser comme troisième paramètre ;

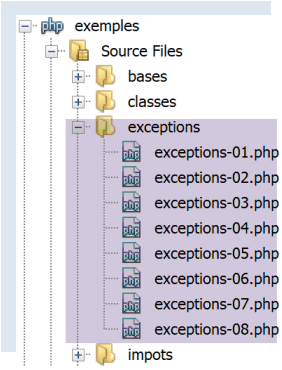
Résultats

```
1   107
2   201
```

1.7 Les exceptions et erreurs

Lorsqu'une méthode d'une classe rencontre une erreur irrécupérable (fichier inexistant, base de données non connectée, connexion réseau hors service), elle n'affiche pas une erreur sur une console (fichier, base de données) mais **lance une exception**. Toutes les exceptions étendent la classe **[\Exception]**. Outre des exceptions, le fonctionnement interne de PHP émet également des erreurs dont la classe de base est la classe **[\Error]**. Les deux classes implémentent l'interface PHP **[\Throwable]**.

1.7.1 L'arborescence des scripts



1.7.2 L'interface `[\Throwable]`

L'interface **`[\Throwable]`** est la suivante :

Sommaire de l'Interface

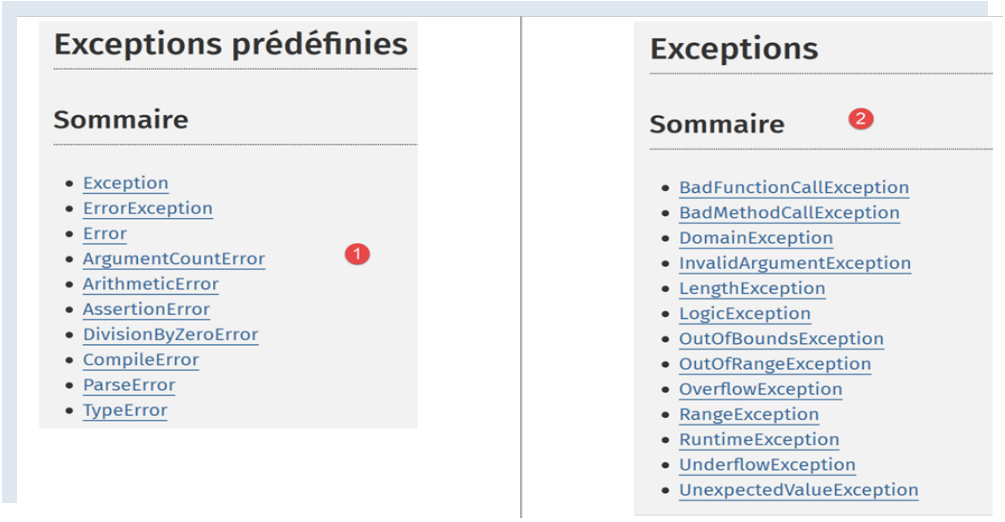
```
Throwable {

   /* Méthodes */
   abstract public getMessage ( void ) : string
   abstract public getCode ( void ) : int
   abstract public getFile ( void ) : string
   abstract public getLine ( void ) : int
   abstract public getTrace ( void ) : array
   abstract public getTraceAsString ( void ) : string
   abstract public getPrevious ( void ) : Throwable
   abstract public __toString ( void ) : string
}
```

Le rôle des méthodes de l'interface est le suivant :

- Throwable::getMessage — Récupère le message
- Throwable::getCode — Récupère le code de l'exception
- Throwable::getFile — Récupère le fichier où l'objet a été crée
- Throwable::getLine — Récupère le numéro de ligne où l'object instancié a été jeté
- Throwable::getTrace — Récupère la trace d'appels
- Throwable::getTraceAsString — Récupère la trace d'appels en tant que chaîne de charactères
- Throwable::getPrevious — Retourne le Throwable précédent
- Throwable::__toString — Récupère la représentation sous forme de chaîne de charactère de l'objet jeté

1.7.3 Les exceptions prédéfinies dans PHP 7

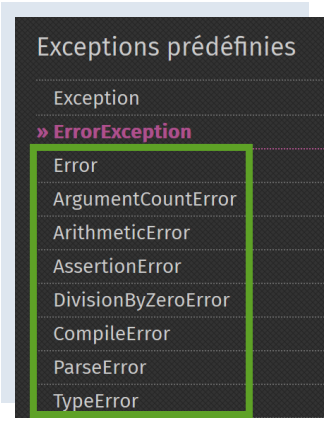
PHP 7 définit plusieurs classes d'exceptions :



- en **[1]**, les exceptions prédéfinies dans PHP ;
- en **[2]**, les exceptions de la bibliothèque SPL (Standard PHP Library) de PHP 7. La bibliothèque SPL est une collection de classes et d'interfaces destinées à résoudre des problèmes rencontrés fréquemment par les développeurs.

1.7.4 Les erreurs prédéfinies dans PHP 7

PHP 7 définit plusieurs classes d'erreurs :



La classe **[\Error]** est la classe parent de toutes les erreurs prédéfinies dans PHP. La classe **[ErrorException]** permet d'encapsuler une instance de la classe **[\Error]** dans une instance de la classe **[\Exception]**. Ceci permet d'uniformiser la gestion des erreurs en ne traitant que des exceptions.

1.7.5 Exemple 1

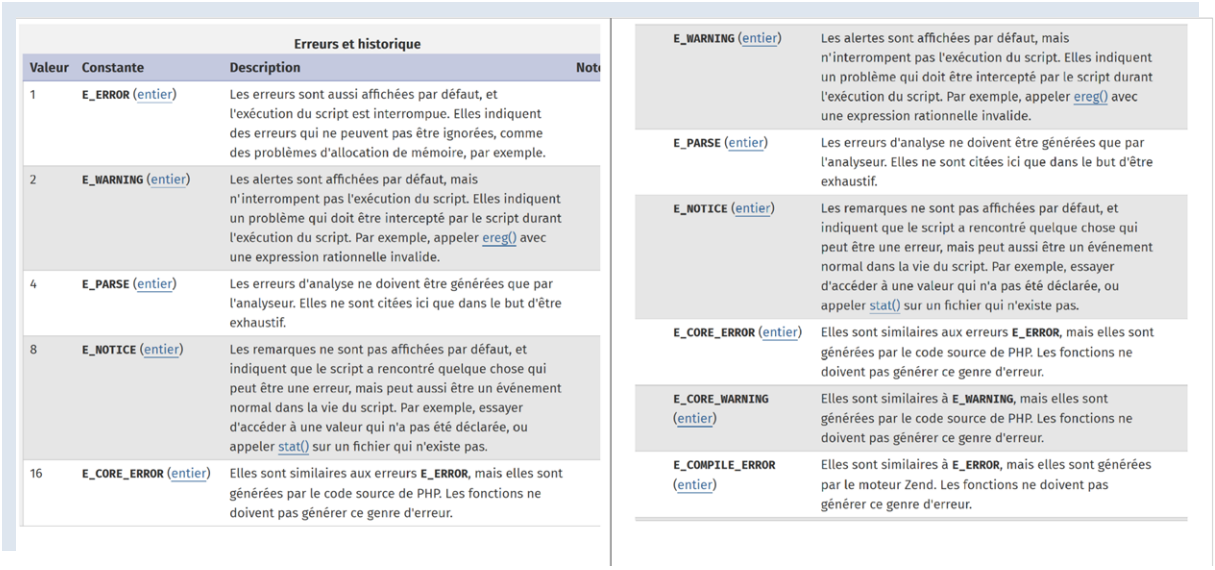
Le premier exemple **[exceptions-01.php]** montre à la fois des erreurs PHP et une exception :

```
1.  <?php
2.
3.  // affichage de toutes les erreurs
4.  ini_set("error_reporting", E_ALL);
5.  ini_set("display_errors", "on");
6.  // code --------
7.  $var=[];
8.  // clé inconnue
9.  print $var["abcd"];
10. // division par zéro
11. $var=7/0;
12. var_dump($var);
13. // tableau à bornes fixes
14. $array = new \SplFixedArray(5);
15. $array[1] = 2;
16. $array[4] = "foo";
```

```
17. // indice en-dehors des bornes
18. $array[5]=8;
```

Commentaires

- ligne 4 : on demande à PHP de signaler toutes les erreurs. Le second paramètre est le niveau d'erreurs demandé :



Erreurs et historique

Valeur	Constante	Description	Not
1	**E_ERROR** (entier)	Les erreurs sont aussi affichées par défaut, et l'exécution du script est interrompue. Elles indiquent des erreurs qui ne peuvent pas être ignorées, comme des problèmes d'allocation de mémoire, par exemple.	
2	**E_WARNING** (entier)	Les alertes sont affichées par défaut, mais n'interrompent pas l'exécution du script. Elles indiquent un problème qui doit être intercepté par le script durant l'exécution du script. Par exemple, appeler ereg() avec une expression rationnelle invalide.	
4	**E_PARSE** (entier)	Les erreurs d'analyse ne doivent être générées que par l'analyseur. Elles ne sont citées ici que dans le but d'être exhaustif.	
8	**E_NOTICE** (entier)	Les remarques ne sont pas affichées par défaut, et indiquent que le script a rencontré quelque chose qui peut être une erreur, mais peut aussi être un événement normal dans la vie du script. Par exemple, essayer d'accéder à une valeur qui n'a pas été déclarée, ou appeler stat() sur un fichier qui n'existe pas.	
16	**E_CORE_ERROR** (entier)	Elles sont similaires aux erreurs **E_ERROR**, mais elles sont générées par le code source de PHP. Les fonctions ne doivent pas générer ce genre d'erreur.	

E_WARNING (entier)	Les alertes sont affichées par défaut, mais n'interrompent pas l'exécution du script. Elles indiquent un problème qui doit être intercepté par le script durant l'exécution du script. Par exemple, appeler ereg() avec une expression rationnelle invalide.
E_PARSE (entier)	Les erreurs d'analyse ne doivent être générées que par l'analyseur. Elles ne sont citées ici que dans le but d'être exhaustif.
E_NOTICE (entier)	Les remarques ne sont pas affichées par défaut, et indiquent que le script a rencontré quelque chose qui peut être une erreur, mais peut aussi être un événement normal dans la vie du script. Par exemple, essayer d'accéder à une valeur qui n'a pas été déclarée, ou appeler stat() sur un fichier qui n'existe pas.
E_CORE_ERROR (entier)	Elles sont similaires aux erreurs **E_ERROR**, mais elles sont générées par le code source de PHP. Les fonctions ne doivent pas générer ce genre d'erreur.
E_CORE_WARNING (entier)	Elles sont similaires à **E_WARNING**, mais elles sont générées par le code source de PHP. Les fonctions ne doivent pas générer ce genre d'erreur.
E_COMPILE_ERROR (entier)	Elles sont similaires à **E_ERROR**, mais elles sont générées par le moteur Zend. Les fonctions ne doivent pas générer ce genre d'erreur.

E_COMPILE_WARNING (entier)	Elles sont similaires à **E_WARNING**, mais elles sont générées par le moteur Zend. Les fonctions ne doivent pas générer ce genre d'erreur.
E_USER_ERROR (entier)	Message d'erreur généré par l'utilisateur. Comparable à **E_ERROR**. Elle est générée par le programmeur en PHP par l'utilisation de la fonction trigger_error(). Les fonctions de PHP ne doivent pas générer ce genre d'erreur.
E_USER_WARNING (entier)	Message d'erreur généré par l'utilisateur. Comparable à **E_WARNING**. Elle est générée par le programmeur en PHP par l'utilisation de la fonction trigger_error(). Les fonctions de PHP ne doivent pas générer ce genre d'erreur.
E_USER_NOTICE (entier)	Message d'erreur généré par l'utilisateur. Comparable à **E_NOTICE**. Elle est générée par le programmeur en PHP par l'utilisation de la fonction trigger_error(). Les fonctions de PHP ne doivent pas générer ce genre d'erreur.

E_RECOVERABLE_ERROR (entier)	Erreur fatale qui peut être captée. Ceci indique qu'une erreur probablement dangereuse s'est produite, mais n'a pas laissé le moteur Zend dans un état instable. Si l'erreur n'est pas attrapée par un gestionnaire d'erreur défini par l'utilisateur (voyez aussi set_error_handler(), l'application arrête prématurément comme si cela était une **E_ERROR**.
E_DEPRECATED (entier)	Alertes d'exécution. Activer cette option pour recevoir des alertes sur les portions de votre code qui pourraient ne pas fonctionner avec les futures versions.
E_USER_DEPRECATED (entier)	Message d'alerte généré par l'utilisateur. Fonctionne de la même façon que **E_DEPRECATED**, mise à part que le message est généré par votre code PHP en utilisant la fonction trigger_error().
E_ALL (entier)	Toutes les erreurs et alertes supportées sauf le niveau **E_STRICT** avant la version 5.4.0 de PHP.

- ligne 5 : on demande d'afficher les erreurs sur la console ;
- ligne 9 : on accède à un élément inexistant du tableau **[$var]** ;
- ligne 11 : on fait une division par zéro ;
- ligne 14 : on crée une instance de la classe **[SplFixedArray]**. Cette classe permet de créer un tableau à bornes fixes et à indices entiers ;
- ligne 18 : on accède à un élément inexistant du tableau ;

Résultats

```
1   Notice: Undefined index: abcd in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-
    01.php on line 9
2
3   Warning: Division by zero in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-01.php
    on line 11
4   float(INF)
5
6   Fatal error: Uncaught RuntimeException: Index invalid or out of range in C:\Data\st-2019\dev\php7\php5-
    exemples\exemples\exceptions\exceptions-01.php:18
7   Stack trace:
8   #0 {main}
9   thrown in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-01.php on line 18
```

Commentaires

- ligne 1 des résultats : accéder à une clé inexistante d'un tableau provoque une erreur PHP de niveau **[E_NOTICE]**. Cela n'interrompt pas l'exécution du script ;
- ligne 3 des résultats : diviser un nombre par zéro provoque une erreur PHP de niveau **[E_WARNING]**. Cela n'interrompt pas l'exécution du script ;
- lignes 6-9 des résultats : accéder à un indice inexistant d'un tableau **[SplFixedArray]** provoque une exception de type **[RuntimeException]** et interrompt l'exécution du script ;

1.7.6 Gérer les exceptions

Le script **[exceptions-02.php]** montre comment gérer les exceptions :

```
1.  <?php
2.
3.  // on affiche toutes les erreurs
4.  ini_set("error_reporting", E_ALL);
5.  ini_set("display_errors", "on");
6.  // on entoure le code par un try / catch
7.  try {
8.    $var = [];
9.    // clé inconnue
10.   print $var["abcd"];
11.   // division par zéro
12.   $var = 7 / 0;
13.   var_dump($var);
14.   // tableau à bornes fixes
15.   $array = new \SplFixedArray(5);
16.   $array[1] = 2;
17.   $array[4] = "foo";
18.   // indice en-dehors des bornes
19.   $array[5] = 8;
20.   // vérification
21.   print "ce message ne sera pas affiché\n";
22. } catch (\Throwable $ex) {
23.   // \Throwable est l'interface implémentée par la plupart des erreurs et exceptions
24.   // on affiche l'exception
25.   print "erreur, message : " . $ex->getMessage() . ", type : " . get_class($ex) . "\n";
26. }
```

Commentaires

- le script est celui qui a été présenté au paragraphe précédent. Seulement maintenant on a entouré le code des lignes 8-19 susceptible de provoquer des erreurs par une structure **try / catch** : si le code des lignes 8-21 provoque (lance) une exception ou une erreur, celle-ci sera gérée par la clause **catch** des lignes 22-26 ;
- ligne 22 : le paramètre de la clause **[catch]** est le type d'exception ou d'erreur que l'on veut gérer. En mettant comme type **[\Throwable]** qui est une interface, on indique qu'on veut gérer toute instance de classe implémentant l'interface **[\Throwable]**. Comme toutes les classes d'erreurs et d'exceptions implémentent cette interface, la clause **[catch]** gère ici toute erreur / exception encapsulée dans une classe ;
- ligne 19 : l'instruction qui déclenche l'erreur et donne naissance à l'exception. Dès qu'il y a exception, il y a branchement sur la clause **[catch]**. Le code derrière la ligne 19 ne sera donc pas exécuté ;

Résultats

```
1   Notice: Undefined index: abcd in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-
    02.php on line 10
2
3   Warning: Division by zero in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-02.php
    on line 12
4   float(INF)
5   erreur, message : Index invalid or out of range, type : RuntimeException
```

Commentaires des résultats

- lignes 1 et 3 : on retrouve les erreurs de niveau **[E_NOTICE]** et **[E_WARNING]**. Ces erreurs ne sont pas des exceptions et ne sont donc pas gérées par la clause **[catch]** ;
- ligne 5 : on a le message d'erreur écrit dans la clause **[catch]**. Il s'est donc produit une exception dérivée de **[\Exception]** ou une erreur dérivée de **[\Error]**. Nous voyons ici qu'il s'agit de la classe **[\RuntimeException]** ;

1.7.7 Paramètres de la clause [catch]

Examinons le script **[exceptions-03.php]** suivant :

```
1.  <?php
2.
3.  // on affiche toutes les erreurs
4.  ini_set("error_reporting", E_ALL);
5.  ini_set("display_errors", "on");
6.
7.  // un tableau à bornes fixes
8.  $array = new \SplFixedArray(5);
9.  try {
10.   // indice en-dehors des bornes
11.   $array[5] = 8;
12. } catch (\Throwable $ex) {
13.   // affichage message d'erreur
14.   print "Erreur 1 : " . $ex->getMessage() . "\n";
15. }
16.
17. try {
18.   // indice en-dehors des bornes
19.   $array[5] = 8;
20. } catch (\Exception $ex) {
21.   // affichage message d'erreur
22.   print "Erreur 2 : " . $ex->getMessage() . "\n";
23. }
24.
25. try {
26.   // indice en-dehors des bornes
27.   $array[5] = 8;
28. } catch (\RuntimeException $ex) {
29.   // affichage message d'erreur
30.   print "Erreur 3 : " . $ex->getMessage() . "\n";
31. }
32. try {
33.   // division par 0
34.   intdiv(5, 0);
35. } catch (\Throwable $ex) {
36.   // affichage message d'erreur
37.   print "Erreur 4 : " . $ex->getMessage() . "\n";
38. }
39.
40. try {
41.   // division par 0
42.   intdiv(5, 0);
43. } catch (\DivisionByZeroError $ex) {
44.   // affichage message d'erreur
45.   print "Erreur 5 : " . $ex->getMessage() . "\n";
46. }
47.
48. try {
49.   // division par 0
50.   intdiv(5, 0);
51. } catch (\Error $ex) {
52.   // affichage message d'erreur
53.   print "Erreur 6 : " . $ex->getMessage() . "\n";
54. }
55.
56. try {
57.   // division par 0
58.   intdiv(5, 0);
59. } catch (\Exception $ex) {
60.   // affichage message d'erreur
61.   print "Erreur 6 : " . $ex->getMessage() . "\n";
62. }
```

Commentaires

- lignes 8-31 : 3 façons différentes de gérer l'exception générée par l'utilisation d'un indice incorrect avec la classe **[\SplFixedArray]**. Nous avons vu que cette erreur générait une exception de type **[RuntimeException]** ;
  - ligne 12 : gère une erreur de type **[\Throwable]**. C'est valide puisque le type **[RuntimeException]** dérive du type **[\Exception]** qui implémente l'interface **[\Throwable]** ;
  - ligne 20 : gère une erreur de type **[\Exception]**. C'est valide puisque le type **[RuntimeException]** dérive du type **[\Exception]** ;

    - ligne 28 : gère une erreur de type **[\RuntimeException]**. C'est la méthode à privilégier puisque c'est le type exact de l'exception générée ;
- lignes 32-62 : 4 façons différentes de gérer l'exception générée par la fonction **[intdiv]** lorsqu'on passe à celle-ci un diviseur égal à 0. La fonction **[intdiv (int $dividend , int $divisor) : int]** fait la division entière $dividend / $divisor. Lorsque le diviseur est nul, l'exception **[\DivisionByzeroError]** est lancée ;
    - ligne 35 : on intercepte toute erreur implémentant l'interface **[\Throwable]**. C'est valide ;
    - ligne 43 : on intercepte le type exact de l'erreur : c'est la méthode à privilégier ;
    - ligne 51 : on intercepte le type **[\Error]**. C'est valide puisque la classe **[DivisionByzeroError]** étend la classe **[Error]** ;
    - ligne 59 : on intercepte le type **[\Exception]**. C'est invalide car la classe **[DivisionByzeroError]** n'a aucun lien avec la classe **[\Exception]** ;

Résultats

```
1   Erreur 1 : Index invalid or out of range
2   Erreur 2 : Index invalid or out of range
3   Erreur 3 : Index invalid or out of range
4   Erreur 4 : Division by zero
5   Erreur 5 : Division by zero
6   Erreur 6 : Division by zero
7
8   Fatal error: Uncaught DivisionByZeroError: Division by zero in C:\Data\st-2019\dev\php7\php5-
    exemples\exemples\exceptions\exceptions-03.php:58
9   Stack trace:
10  #0 C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-03.php(58): intdiv(5, 0)
11  #1 {main}
12  thrown in C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-03.php on line 58
```

1.7.8 Clause [finally]

La structure try / catch peut avoir un troisième élément et devenir une structure **try / catch / finally**. Le code de la clausse **[finally]** est exécuté dans les deux cas suivants :

- la clause **[try]** ne lance pas d'exception. Elle est alors exécutée entièrement puis l'exécution du code passe à la clause **[finally]** qui est exécutée entièrement ;
- la clause **[try]** lance une exception. Elle est alors exécutée jusqu'à l'instruction qui lance l'exception. L'exécution du code passe alors à la clause **[catch]** qui est exécutée entièrement. Puis l'exécution du code passe à la clause **[finally]** qui est exécutée entièrement ;

Finalement, le code de la clause **[finally]** est toujours exécuté. Ce scénario est utile dans le cas suivant :

- dans le **[try]**, le code a obtenu des ressources (fichiers, bases de données, connexions réseau, files d'attente). En général ces ressources sont coûteuses en mémoire. Il faut alors les rendre (on dit le plus souvent **fermer**) dès que c'est possible ;
- si l'acquisition des ressources a été faite dans le **[try]**, on mettra leur restitution dans le **[finally]**. Cela nous assure que dans tous les cas (erreur ou pas), les ressources acquises sont restituées au système ;

Le script suivant **[exemples/exceptions/exceptions-04.php]** nous montre le fonctionnement de la clause **[finally]** dans diverses situations :

```
1.  <?php
2.
3.  // ou crée une instance d'exception
4.  $e = new \Exception("Erreur...");
5.  var_dump($e);
6.
7.  // premier test
8.  try {
9.    print "Premier test\n";
10.   throw $e;
11. } catch (\Exception $ex1) {
12.   // msg d'erreur
13.   print $ex1->getMessage() . "\n";
14. } finally {
15.   print "Terminé\n";
16. }
17.
18. // second test
19. try {
20.   print "Second test\n";
21. } catch (\Exception $ex1) {
22.   print $ex1->getMessage() . "\n";
```

```
23. } finally {
24.   print "Terminé\n";
25. }
26.
27. // troisième test
28. try {
29.   print "Troisième test\n";
30.   return;
31. } catch (\Exception $ex1) {
32.   print $ex1->getMessage() . "\n";
33. } finally {
34.   print "Terminé\n";
35. }
```

Commentaires du code

- ligne 4 : $e est une instance de la classe prédéfinie **[\Exception]**. On va la lancer à différents endroits ;
- lignes 8-16 : l'exception $e est lancée dans le **[try]** (ligne 10) ;
- ligne 11 : l'exception **[\Exception]** est interceptée et son message d'erreur écrit sur la console ;
- lignes 14-16 : la clause **[finally]** écrit un message. D'après ce qui a été dit précédemment, ce message devrait être tout le temps écrit, erreur ou pas dans le **[try]** ;
- lignes 19-25 : il n'y a pas d'erreur dans le **[try]**. On devrait là également passer dans le **[finally]** ;
- lignes 28-35 : il y a une instruction **[return]** dans le try et pas d'erreur. On peut se demander alors si on va passer dans la clause **[finally]**. L'exécution montre que oui ;

Résultats

```
1   Premier test
2   Erreur…
3   Terminé
4   Second test
5   Terminé
6   Troisième test
7   Terminé
```

Examinons un autre cas **[exceptions-05.php]** :

```
1.  <?php
2.
3.  // quatrième test
4.  try {
5.    print "Quatrième test\n";
6.    exit;
7.  } finally {
8.    print "Terminé\n";
9.  }
```

Commentaires

- ligne 6 : l'instruction **[exit]** arrête immédiatement l'exécution du script : la clause **[finally]** n'est pas exécutée ;
- lignes 4-9 : un exemple de try / catch / finally sans clause **[catch]**. C'est possible ;

Résultats

```
1   Quatrième test
```

1.7.9 Créer ses propres classes d'exceptions

Dans un projet un peu important, il est utile de différentier les différentes erreurs en les encapsulant dans différentes classes d'exceptions. Dans le script précédent, nous avons vu que toute exception pouvait être interceptée par une clause **[catch (\Throwable]**. C'est recommandé si on n'a aucune idée de l'erreur interceptée et que le traitement est le même pour toutes les erreurs. C'est parfois le cas mais on a souvent besoin d'adapter le traitement au type exact de l'erreur. Il faut alors différentier les erreurs entre-elles.

Examinons le script **[exceptions-06.php]** suivant :

```
1.  <?php
2.
3.  // on définit notre propre famille d'exceptions
4.  class Exception1 extends \RuntimeException {
```

```
5.
6.  }
7.
8.  class Exception2 extends \RuntimeException {
9.
10. }
11.
12. // ou utilise nos exceptions
13. $e1 = new Exception1("Erreur1...");
14. var_dump($e1);
15. $e2 = new Exception2("Erreur2...");
16. var_dump($e2);
17.
18. // premier test
19. print ("premier test\n");
20. try {
21.   // on lance un type Exception1
22.   throw $e1;
23. } catch (Exception1 $ex1) {
24.   print "Exception 1" . "\n";
25.   print $ex1->getMessage() . "\n";
26. } catch (Exception2 $ex2) {
27.   print "Exception 2" . "\n";
28.   print $ex2->getMessage() . "\n";
29. }
30.
31. // second test
32. print ("second test\n");
33. try {
34.   // on lance un type Exception2
35.   throw $e2;
36. } catch (Exception1 $ex1) {
37.   print "Exception 1" . "\n";
38.   print $ex1->getMessage() . "\n";
39. } catch (Exception2 $ex2) {
40.   print "Exception 2" . "\n";
41.   print $ex2->getMessage() . "\n";
42. }
43.
44. // troisième test
45. print ("troisième test\n");
46. try {
47.   // on lance un type Exception1
48.   throw $e1;
49. } catch (Exception1 | Exception2 $ex) {
50.   print "Exception 1 ou 2" . "\n";
51.   print $ex->getMessage() . "\n";
52. }
53.
54. // quatrième test
55. print ("quatrième test\n");
56. try {
57.   // on lance un type Exception2
58.   throw $e2;
59. } catch (Exception1 | Exception2 $ex) {
60.   print "Exception 1 ou 2" . "\n";
61.   print $ex->getMessage() . "\n";
62. }
```

Commentaires

- lignes 4-10 : on définit deux classes **[Exception1]** et **[Exception2]** toutes deux dérivées de la classe prédéfinie **[\RuntimeException]**. Le corps de ces classes est vide. Autrement dit, on ne les utilise que pour leurs types : c'est parce qu'elles ont des types différents qu'on va pouvoir différentier ces deux exceptions dans les clauses **[catch]** ;
- lignes 13-16 : on définit deux variables $e1 et $e2 ayant respectivement les types **[Exception1]** et **[Exception2]** ;
- lignes 20-29 : on a une structure try / catch / catch. Cela permet de gérer différents types d'exception avec différentes clauses **[catch]** ;
- ligne 23 : intercepte les exceptions de type **[Exception1]** ;
- ligne 26 : intercepte les exceptions de type **[Exception2]** ;
- ligne 49 : intercepte les exceptions de type **[Exception1]** ou (|) **[Exception2]** ;

Résultats

```
1   object(Exception1)#1 (7) {
```

```
2     ["message":protected]=>
3     string(10) "Erreur1…"
4     ["string":"Exception":private]=>
5     string(0) ""
6     ["code":protected]=>
7     int(0)
8     ["file":protected]=>
9     string(76) "C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-06.php"
10    ["line":protected]=>
11    int(13)
12    ["trace":"Exception":private]=>
13    array(0) {
14    }
15    ["previous":"Exception":private]=>
16    NULL
17  }
18  object(Exception2)#2 (7) {
19    ["message":protected]=>
20    string(10) "Erreur2…"
21    ["string":"Exception":private]=>
22    string(0) ""
23    ["code":protected]=>
24    int(0)
25    ["file":protected]=>
26    string(76) "C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-06.php"
27    ["line":protected]=>
28    int(15)
29    ["trace":"Exception":private]=>
30    array(0) {
31    }
32    ["previous":"Exception":private]=>
33    NULL
34  }
35  premier test
36  Exception 1
37  Erreur1…
38  second test
39  Exception 2
40  Erreur2…
41  troisième test
42  Exception 1 ou 2
43  Erreur1…
44  quatrième test
45  Exception 1 ou 2
46  Erreur2…
```

Commentaires des résultats

- lignes 1-17 : le 'contenu' d'une exception :
  - lignes 2-3 : le message d'erreur ;
  - lignes 6-7 : le code d'erreur ;
  - lignes 8-9 : le nom du fichier dans lequel s'est produite l'exception ;
  - lignes 10-11 : la ligne à laquelle s'est produite l'exception ;
  - lignes 15-16 : l'exception précédente. Une exception peut encapsuler une autre exception et définir ainsi une pile d'exceptions. L'attribut **`[previous]`** va permettre d'exploiter cette pile ;

1.7.10 Relancer une exception

Une exception peut être lancée plusieurs fois comme le montre le script **`[exceptions-07.php]`** suivant :

```
1.  <?php
2.
3.  try {
4.    try {
5.      // on lance une exception
6.      throw new \Exception("test");
7.    } catch (\Exception $ex) {
8.      // on relance l'exception interceptée
9.      throw $ex;
10.   } finally {
11.     // on passera bien dans le finally
12.     print "finally 1\n";
13.   }
14. } catch (\Exception $ex2) {
```

```
15.   // on récupère bien l'exception initiale
16.   print $ex2->getMessage() . " dans try / catch / finally externe\n";
17. } finally {
18.   // on passera bien dans le finally
19.   print "finally 2\n";
20. }
```

Commentaires

- ligne 6 : on lance une exception ;
- ligne 7 : on l'intercepte ;
- ligne 9 : on la relance. Elle passe alors dans la structure try / catch / finally du niveau supérieur ;
- ligne 14 : on l'intercepte de nouveau ;
- lignes 10-12 : l'exécution montre que même après le **[throw]** de la ligne 9, on passe bien dans la clause **[finally]** du try / catch / finally ;

Résultats

```
1   finally 1
2   test dans try / catch / finally externe
3   finally 2
```

1.7.11 Exploitation d'une pile d'exceptions

Une exception peut encapsuler une autre exception qui elle-même peut en encapsuler une autre formant finalement une pile d'exceptions. Voici un exemple **[exceptions-08.php]** :

```
1.  <?php
2.
3.  // on définit notre propre famille d'exceptions
4.  class Exception1 extends \RuntimeException {
5.
6.  }
7.
8.  class Exception2 extends \RuntimeException {
9.
10. }
11.
12. class Exception3 extends \RuntimeException {
13.
14. }
15.
16. // ou utilise nos exceptions
17. $e1 = new Exception1("Erreur 1...", 1, new Exception2("Erreur 2...", 2, new Exception3("Erreur 3...")));
18. var_dump($e1);
19. // exploitation de l'exception courante
20. print $e1->getMessage() . "\n";
21. $e = $e1;
22. while ($e->getPrevious() !== NULL) {
23.   // exception précédente
24.   $e = $e->getPrevious();
25.   // message d'erreur
26.   print $e->getMessage() . "\n";
27. }
```

Commentaires

- lignes 4-14 : définissent trois classes d'exceptions dérivées de l'exception prédéfinie **[RuntimeException]** ;
- ligne 17 : une instance de la classe **[Exception3]** est encapsulée dans une instance de la classe **[Exception2]** qui elle-même est encapsulée dans une instance de la classe **[Exception1]**. Le constructeur utilisé ici est le constructeur de la classe **[Exception]** :

```
public __construct ([ string $message = "" [, int $code = 0 [, Throwable $previous = NULL
]]] )
```

Le 3e paramètre du constructeur permet d'encapsuler une exception. Cela peut être utile dans le scénario suivant :

- on définit une méthode M qui peut générer une exception de type **[Exception1]** et seulement de ce type pour des raisons de compatibilité par exemple avec une interface ;

- or dans la méthode M peuvent se produire d'autres types d'exceptions. Pour remonter une erreur au code appelant la méthode M, on encapsulera alors ces exceptions dans le type **[Exception1]** que l'on lancera. Cela permet de ne pas perdre l'information contenue dans l'exception encapsulée et qui a été la cause originelle de l'erreur ;

- les lignes 20-27 montrent comment gérer la pile des exceptions internes à une exception ;

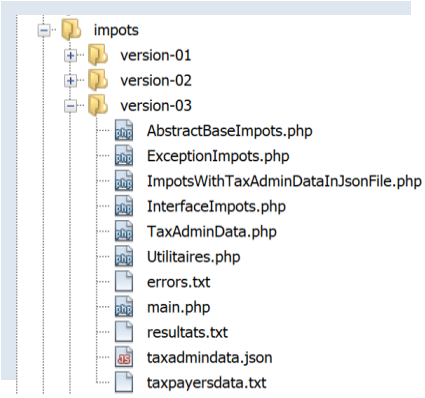
Résultats

```
1   object(Exception1)#1 (7) {
2     ["message":protected]=>
3     string(11) "Erreur 1…"
4     ["string":"Exception":private]=>
5     string(0) ""
6     ["code":protected]=>
7     int(1)
8     ["file":protected]=>
9     string(76) "C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-08.php"
10    ["line":protected]=>
11    int(17)
12    ["trace":"Exception":private]=>
13    array(0) {
14    }
15    ["previous":"Exception":private]=>
16    object(Exception2)#2 (7) {
17      ["message":protected]=>
18      string(11) "Erreur 2…"
19      ["string":"Exception":private]=>
20      string(0) ""
21      ["code":protected]=>
22      int(2)
23      ["file":protected]=>
24      string(76) "C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-08.php"
25      ["line":protected]=>
26      int(17)
27      ["trace":"Exception":private]=>
28      array(0) {
29      }
30      ["previous":"Exception":private]=>
31      object(Exception3)#3 (7) {
32        ["message":protected]=>
33        string(11) "Erreur 3…"
34        ["string":"Exception":private]=>
35        string(0) ""
36        ["code":protected]=>
37        int(0)
38        ["file":protected]=>
39        string(76) "C:\Data\st-2019\dev\php7\php5-exemples\exemples\exceptions\exceptions-08.php"
40        ["line":protected]=>
41        int(17)
42        ["trace":"Exception":private]=>
43        array(0) {
44        }
45        ["previous":"Exception":private]=>
46        NULL
47      }
48    }
49  }
50  Erreur 1…
51  Erreur 2…
52  Erreur 3…
```

1.8 Exercice d'application – version 3

On reprend l'exercice déjà étudié précédemment (pages 60 et 66) pour le résoudre avec un code PHP utilisant une classe.

1.8.1 L'arborescence des scripts



1.8.2 L'exception `[ExceptionImpots]`

Dans la version 03, lorsqu'un constructeur ou une méthode de classe rencontrera une erreur, elle lancera une exception de type **`[ExceptionImpots]`** suivant :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  class ExceptionImpots extends \RuntimeException {
7.
8.    public function __construct(string $message, int $code = 0) {
9.      parent::__construct($message, $code);
10.   }
11. }
```

Commentaires

- ligne 4 : la classe **`[ExceptionImpots]`** est dans l'espace de noms **`[Application]`** ;
- ligne 6 : la classe **`[ExceptionImpots]`** étend la classe prédéfinie dans PHP **`[RuntimeException]`** ;
- ligne 8 : le constructeur attend deux paramètres :
  - **$message** : est le message d'erreur associée à l'exception ;
  - **$code** : est le code d'erreur associé à l'exception. S'il n'est pas présent, alors le code 0 sera utilisé ;

1.8.3 La classe `[TaxAdminData]`

Dans la version 02, les données de l'administration fiscale ont été rassemblées :

- d'abord dans un fichier jSON ;
- puis de ce fichier jSON à un tableau associatif ;

Dans la version 03, les données de l'administration fiscale sont toujours dans le fichier **`[taxadmindata.json]`** mais avec des noms d'attribut différents :

```
1   {
2           "limites": [
3                   9964,
4                   27519,
5                   73779,
6                   156244,
7                   0
8           ],
9           "coeffR": [
10                  0,
11                  0.14,
12                  0.3,
13                  0.41,
14                  0.45
15          ],
16          "coeffN": [
17                  0,
```

```
                1394.96,
                5798,
                13913.69,
                20163.45
            ],
            "plafondQfDemiPart": 1551,
            "plafondRevenusCelibatairePourReduction": 21037,
            "plafondRevenusCouplePourReduction": 42074,
            "valeurReducDemiPart": 3797,
            "plafondDecoteCelibataire": 1196,
            "plafondDecoteCouple": 1970,
            "plafondImpotCouplePourDecote": 2627,
            "plafondImpotCelibatairePourDecote": 1595,
            "abattementDixPourcentMax": 12502,
            "abattementDixPourcentMin": 437
    }
```

Dans la version 02, ce fichier servait à initialiser un tableau associatif. Dans la version 03 le fichier va initialiser la classe **[TaxAdminData]** suivante :

```php
<?php

namespace Application;

class TaxAdminData {
  // tranches d'impôt
  private $limites;
  private $coeffR;
  private $coeffN;
  // constantes de calcul de l'impôt
  private $plafondQfDemiPart;
  private $plafondRevenusCelibatairePourReduction;
  private $plafondRevenusCouplePourReduction;
  private $valeurReducDemiPart;
  private $plafondDecoteCelibataire;
  private $plafondDecoteCouple;
  private $plafondImpotCouplePourDecote;
  private $plafondImpotCelibatairePourDecote;
  private $abattementDixPourcentMax;
  private $abattementDixPourcentMin;

  // initialisation
  public function setFromJsonFile(string $taxAdminDataFilename): TaxAdminData {
    // on récupère le contenu du fichier des données fiscales
    $fileContents = \file_get_contents($taxAdminDataFilename);
    $erreur = FALSE;
    // erreur ?
    if (!$fileContents) {
      // on note l'erreur
      $erreur = TRUE;
      $message = "Le fichier des données [$taxAdminDataFilename] n'existe pas";
    }
    if (!$erreur) {
      // on récupère le code JSON du fichier de configuration dans un tableau associatif
      $arrayTaxAdminData = \json_decode($fileContents, true);
      // erreur ?
      if ($arrayTaxAdminData === FALSE) {
        // on note l'erreur
        $erreur = TRUE;
        $message = "Le fichier de données JSON [$taxAdminDataFilename] n'a pu être exploité correctement";
      }
    }
    // erreur ?
    if ($erreur) {
      // on lance une exception
      throw new ExceptionImpots($message);
    }
    // initialisation des attributs de la classe
    foreach ($arrayTaxAdminData as $key => $value) {
      $this->$key = $value;
    }
    // on vérifie que toutes les clés ont été initialisées
    $arrayOfAttributes = \get_object_vars($this);
    foreach ($arrayOfAttributes as $key => $value) {
      if (!isset($this->$key)) {
        throw new ExceptionImpots("L'attribut [$key] de [TaxAdminData] n'a pas été initialisé");
```

```
57.       }
58.     }
59.     // on vérifie qu'on a que des valeurs réelles
60.     foreach ($this as $key => $value) {
61.       // $value doit être un nbre réel >=0 ou un tableau de réels >=0
62.       $result = $this->check($value);
63.       // erreur ?
64.       if ($result->erreur) {
65.         // on lance une exception
66.         throw new ExceptionImpots("La valeur de l'attribut [$key] est invalide");
67.       } else {
68.         // on note la valeur
69.         $this->$key = $result->value;
70.       }
71.     }
72.     // on rend l'objet
73.     return $this;
74.   }
75.
76.   private function check($value): \stdClass {
77.     ….
78.     return $result;
79.   }
80.
81.   // toString
82.   public function __toString() {
83.     // chaîne Json de l'objet
84.     return \json_encode(\get_object_vars($this), JSON_UNESCAPED_UNICODE);
85.   }
86.
87.   // getters et setters
88.   public function getLimites() {
89.     return $this->limites;
90.   }
91.
92.   public function getCoeffR() {
93.     return $this->coeffR;
94.   }
95.
96.   …
97.
98.
99.   public function setLimites($limites) {
100.    $this->limites = $limites;
101.    return $this;
102.  }
103.
104.  public function setCoeffR($coeffR) {
105.    $this->coeffR = $coeffR;
106.    return $this;
107.  }
108.…
109.
110.}
```

Commentaires

- lignes 6-20 : les attributs qui vont accueillir les attributs de même nom du fichier jSON **[taxadmindata.json]**. C'est un point important : les attributs de la classe **[TaxAdminData]** sont **identiques** à ceux du fichier jSON **[taxadmindata.json]**. Cette particularité facilite beaucoup l'écriture du code ;
- la classe **[TaxAdminData]** n'a pas de constructeur. En PHP, il n'est pas possible d'avoir plusieurs constructeurs. En fixer un empêche alors d'initialiser l'objet d'une autre façon. Dans la suite, nos classes n'auront pas de constructeur mais plusieurs méthodes de type **[setFromQqChose]** qui permettront de l'initialiser de différentes façons. La construction d'un objet de type **[TaxAdminData]** se fait alors avec l'expression :

```
(new TaxAdminData())→setFromQqChose(…)
```

- ligne 23 : la méthode **[setFromJsonFile]** initialise les attributs de la classe avec ceux de même nom dans le fichier **[$jsonFilename]** ;
- lignes 24-42 : le fichier jSON est exploité pour construire le tableau associatif **[$arrayTaxAdminData]**. Nous avons déjà rencontré ce code dans le script **[main.php]** de la version 02 ;
- lignes 44-47 : si on a rencontré une erreur dans l'exploitation du fichier jSON, on lance une exception. Celle-ci remontera jusqu'au script principal **[main.php]** ;

- lignes 48-51 : les attributs de la classe sont initialisés. On profite ici du fait que le tableau associatif **[$arrayTaxAdminData]** et la classe **[TaxAdminData]** ont des attributs de mêmes noms que les valeurs provenant du fichier jSON ;
- lignes 53-57 : on vérifie que tous les attributs de la classe **[TaxAdminData]** ont été initialisés ;
- ligne 53 : l'expression **[get_object_vars($this)]** rend un tableau associatif dont les attributs sont ceux de l'objet **[$this]**, donc les attributs de la classe **[TaxAdminData]**. Ici il faut comprendre que l'opération d'initialisation des lignes 48-51 a pu ajouter des attributs à l'objet **[$this]**. Ainsi si on écrit :

```
$this->x = "1000";
```

  alors l'attribut **[x]** est ajouté à l'objet **[$this]** même si cet attribut n'a pas été déclaré dans la classe **[TaxAdminData]**. Ce qui est sûr, c'est que les attributs des lignes 6-20 font bien partie de l'objet **[$this]**, mais ils ont pu être non initialisés. C'est une erreur facile à faire, il suffit de se tromper dans un nom d'attribut dans le fichier **[taxadmindata.json]** ;
- lignes 54-57 : on passe en revue tous les attributs de **[$this]** et si l'un d'eux n'a pas été initialisé, on lance une exception ;
- un attribut peut être initialisé avec une valeur incorrecte. En PHP, il n'est pas possible de donner un type aux attributs. Ainsi l'opération :

```
$this→plafondQfDemiPart=’abcd’
```

  est possible alors que l'attribut **[$plafondQfDemiPart]** devrait être réel ;
- lignes 59-71 : on vérifie que chacun des attributs de la classe a une valeur numérique réelle positive ou nulle. C'est la fonction **[check]** de la ligne 76 qui fait ce travail. Son paramètre **[$value]** est soit une unique valeur soit un tableau de valeurs ;
- ligne 62 : la fonction **[check]** rend un objet de type **[\stdClass]** avec deux attributs :
  - **[erreur]** : à TRUE s'il y a eu erreur, à FALSE sinon ;
  - **[value]** : la valeur numérique réelle correspondant au paramètre **[$value]** passé en paramètre, ligne 62 ;
- ligne 64 : on regarde si la vérification a réussi ou pas ;
- ligne 66 : si un attribut n'est pas un nombre réel positif ou nul, on lance une exception ;
- ligne 69 : sinon on note sa valeur numérique ;
- ligne 73 : on rend l'objet **[$this]** comme résultat ;

La fonction **[check]** est la suivante :

```
1.    private function check($value): \stdClass {
2.      // $value est soit un tableau d'éléments soit un unique élément
3.      // on crée un tableau
4.      if (!\is_array($value)) {
5.        $tableau = [$value];
6.      } else {
7.        $tableau = $value;
8.      }
9.      // on transforme le tableau d'éléments de type non connu en tableau de réels
10.     $newTableau = [];
11.     $result = new \stdClass();
12.     // les éléments du tableau doivent être des nombres décimaux positifs ou nuls
13.     $modèle = '/^\s*([+]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/';
14.     for ($i = 0; $i < count($tableau); $i ++) {
15.       if (preg_match($modèle, $tableau[$i])) {
16.         // on met le float dans newTableau
17.         $newTableau[] = (float) $tableau[$i];
18.       } else {
19.         // on note l'erreur
20.         $result->erreur = TRUE;
21.         // on quitte
22.         return $result;
23.       }
24.     }
25.     // on rend le résultat
26.     $result->erreur = FALSE;
27.     if (!\is_array($value)) {
28.       // une seule valeur
29.       $result->value = $newTableau[0];
30.     } else {
31.       // une liste de valeurs
32.       $result->value = $newTableau;
33.     }
34.     return $result;
35.   }
```

Commentaires

- ligne 1 : le paramètre **[$value]** est soit un tableau soit un unique élément. Par ailleurs, on ne connaît pas son type. La valeur provient du fichier **[taxadmindata.json]**. Selon les valeurs inscrites dans ce fichier, les valeurs lues peuvent être des entiers, des réels, des chaînes, des booléens. Par exemple :

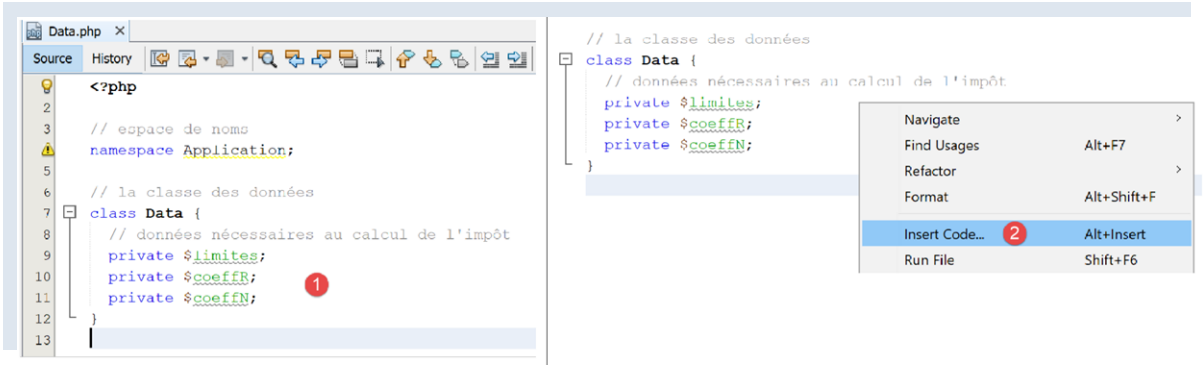
```
1   "plafondQfDemiPart": 1551,
2   "plafondQfDemiPart": 1551.78,
3   "plafondQfDemiPart": "1551",
4   "plafondQfDemiPart": "xx",
```

  Dans le cas 1, la valeur est de type **[entier]**, dans le cas 2 de type **[réel]**, dans le cas 3 de type **[string]** pouvant être converti en nombre, dans le cas 4 de type **[string]** ne pouvant pas être converti en nombre ;
- lignes 4-8 : on crée un tableau à partir du paramètre **[$value]** reçu en paramètre ligne 1 ;
- ligne 10 : le tableau qu'on va remplir avec des nombres réels ;
- ligne 11 : le résultat sera un objet de type **[\stdClass]** ;
- ligne 13 : expression relationnelle d'un nombre réel positif ou nul ;
- lignes 14-24 : on vérifie que tous les éléments du tableau **[$tableau]** sont des nombres réels positifs ou nuls et on remplit le tableau **[$newTableau]** avec ces éléments transformés en type **[float]** (ligne 17) ;
- lignes 18-23 : dès qu'on détecte un élément qui n'est pas un nombre réel positif ou nul, on note l'erreur dans le résultat et on rend celui-ci ;
- lignes 25-34 : cas où tous les éléments du tableau **[$tableau]** ont été déclarés corrects ;
- ligne 32 : la valeur rendue **[$result->value]** est un tableau de réels **[float]** ou un réel unique ;

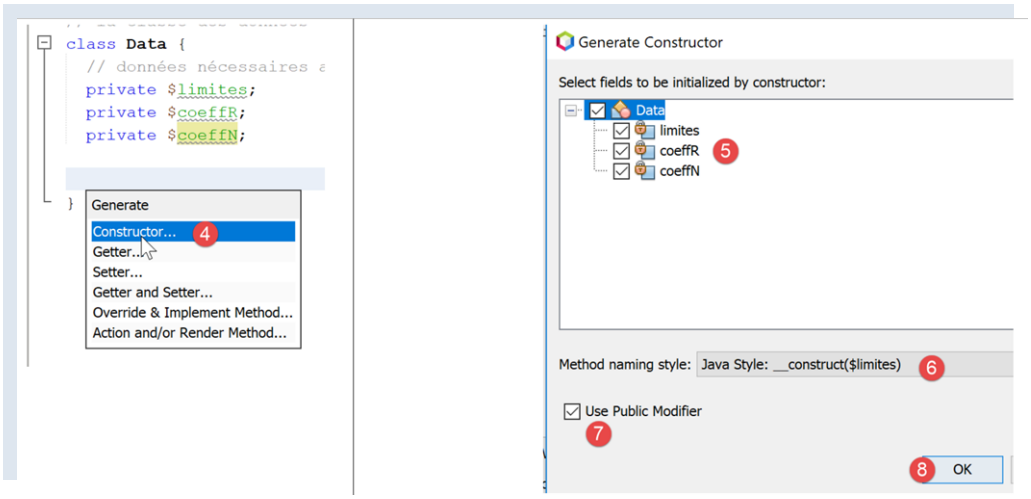
La fonction **[__toString]** des lignes 82-85 rend la chaîne jSON des attributs et valeurs de l'objet **[$this]**.

Lignes 87-110 : les getters et setters de la classe ;

Note : il peut être parfois un peu pénible d'avoir à écrire tous les get / set d'une classe surtout lorsqu'il y a beaucoup d'attributs. Netbeans peut générer automatiquement ceux-ci ainsi que le constructeur. Pour ce faire, mettez simplement les attributs **[1]** :

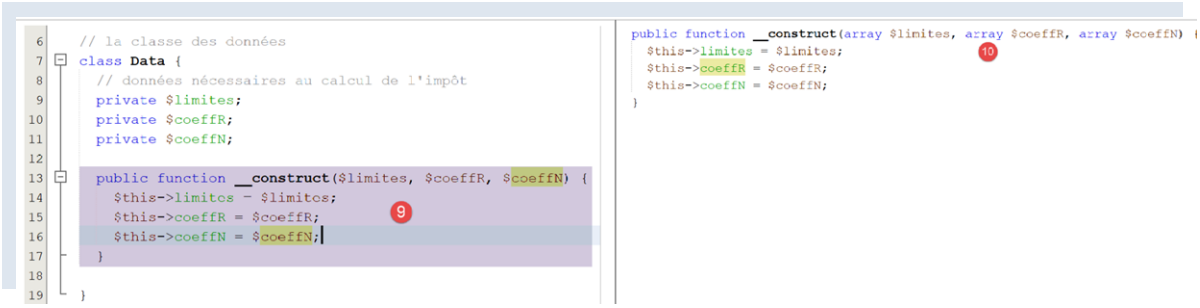


- en **[2]**, cliquez droit là ou voulez insérer du code puis choisissez l'option **[Insert Code]** ;



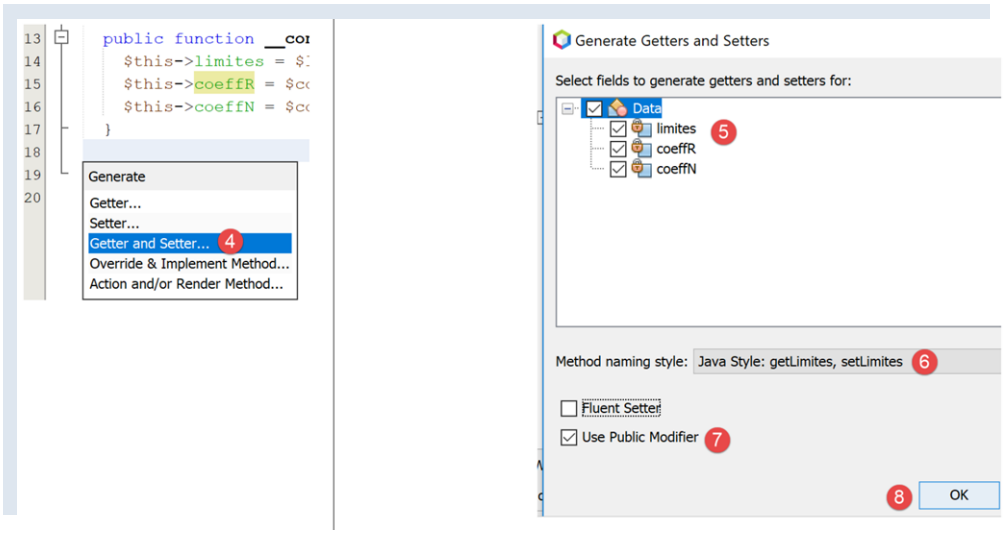
- en **[4]**, indiquez que vous voulez générer le constructeur ;
- en **[5]**, cochez tous les attributs : cela veut dire que vous voulez que le constructeur ait un paramètre pour chacun des attributs ;

- en **[6]**, prenez le style des constructeurs Java ;
- en **[7]**, indiquez que vous voulez explicitement le mot clé **[public]** devant le constructeur ;
- en **[8]**, validez ;

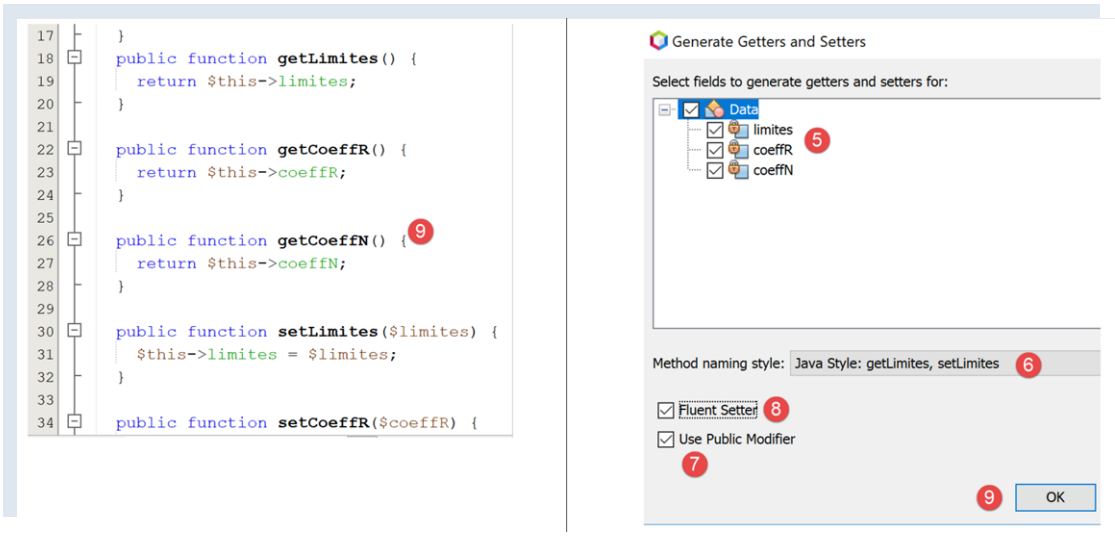


- en **[9]**, Netbeans a généré le constructeur. Cependant il n'a pas pu mettre le type des paramètres parce qu'il ne les connaît pas. Ajoutez-les vous-même **[10]** ;

Pour générer les getters et setters, recommencez les étapes 2-4, et à l'étape 4, choissez **[Getter and Setter]** :



- en **[5]**, indiquez que vous voulez les getters et setters pour chacun des attributs ;
- en **[6]**, indiquez que vous voulez les getters et setters dans le style utilisé par Java : setAttribut, getAttribut ;
- en **[7]**, indiquez que vous que ces getters et setters soient publics ;
- en **[8]**, validez ;

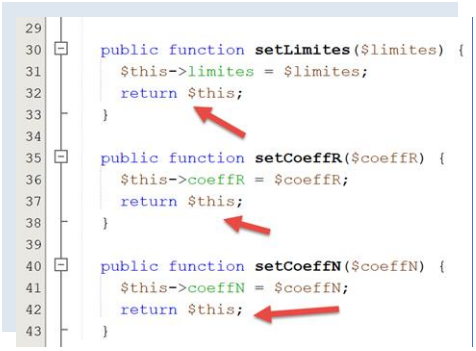


- en **[9]**, les getters et setters générés par Netbeans ;

Effacez ces getters et setters et recommencez les étapes 2-7.

- en **[8]**, cochez l'option **[Fluent Setter]** que nous n'avions pas cochée précédemment ;

Le résultat obtenu est le suivant :



Chaque setter se termine par une opération **[return $this]**. Ceci permet d'initialiser les attributs de la façon suivante :

```
$data→setLimites($limites)→setCoeffR($coeffR)→setCoeffN($coeffN) ;
```

En effet, la valeur de **[$data→setLimites($limites)]** (ligne 32 du code) est **[$this]**, donc ici **[$data]**. On peut donc appeler la méthode **[setCoeffR($coeffR)]** de cet objet et ainsi de suite, puisqu'à son tour, cette méthode rend elle aussi **[$this]** (ligne 37 du code). Cette écriture des méthodes d'une classe qui fait que les méthodes qui ne devraient rien rendre rendent l'objet **[$this]** s'appellent une écriture **fluente**. Elle facilite l'utilisation de ces méthodes.

1.8.4 L'interface [InterfaceImpots]

Nous définissons maintenant l'interface **[InterfaceImpots]** suivante **[InterfaceImpots.php]** :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  interface InterfaceImpots {
7.
8.    // récupérer les données des tranches d'impôt permettant le calcul de l'impôt
9.    // peut lancer l'exception ImpotsException
10.   public function getTaxAdminData(): TaxAdminData;
11.
12.   // l'interface sait calculer un impôt
13.   public function calculerImpot(string $marié, int $enfants, int $salaire): array;
14.
15.   // l'interface sait exploiter des données dans des fichiers texte
16.   // $usersFilename : fichier des données utilisateur sous la forme statut marital, nombre d'enfants, salaire annuel
17.   // $resultsFilename : fichier des des résultats sous la forme statut marital, nombre d'enfants, salaire annuel, montat de
      l'impôt
18.   // $errorsFilename : fichier des erreurs rencontrées
19.   // peut lancer l'exception ExceptionImpots
20.   public function executeBatchImpots(string $usersFileName, string $resultsFileName, string $errorsFileName): void;
21. }
```

Commentaires

- ligne 4 : l'interface est placée dans l'espace de noms **[Application]** ;
- ligne 6 : l'interface permettant le calcul des impôts ;
- ligne 10 : la méthode **[getTaxAdminData]** permettra d'acquérir les données de l'administration fiscale dans un objet de type **[TaxAdminData]** que nous venons de présenter. Comme ces données peuvent être dans un fichier ou une base de données voire sur le réseau, la méthode **[getTaxAdminData]** peut échouer à obtenir les données. Dans ce cas, elle lancera une exception de type **[ExceptionImpots]**. C'est la méthode standard en programmation objet pour signaler une erreur rencontrée dans une méthode ou un constructeur ;
- ligne 13 : la méthode **[calculerImpot]** permettra de calculer l'impôt d'un usager ;
- ligne 20 : la méthode **[executeBatchImpots]** permettra de calculer l'impôt de plusieurs contribuables :
  - **[$usersFileName]** est le nom du fichier texte contenant les données des contribuables ;
  - **[$resultsFileName]** est le nom du fichier texte contenant le montant de l'impôt pour ces contribuables ;

- **[$errorsFileName]** est le nom du fichier texte contenant les erreurs rencontrées lors de l'exploitation de ces fichiers ;

Le contenu du fichier texte **[$usersFileName]** pourrait être le suivant :

```
1   oui,2,55555
2   oui,2,50000
3   oui,3,50000
4   non,2,100000
5   non,3x,100000
6   oui,3,100000
7   oui,5,100000x
8   non,0,100000
9   oui,2,30000
10  non,0,200000
11  oui,3,200000
```

On notera que les lignes 5 et 7 contiennent des éléments erronés.

Le contenu du fichier texte **[$resultsFileName]** sera alors le suivant :

```
1   {"marié":"oui","enfants":2,"salaire":55555,"impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.14}
2   {"marié":"oui","enfants":2,"salaire":50000,"impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.1
    4}
3   {"marié":"oui","enfants":3,"salaire":50000,"impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14}
4   {"marié":"non","enfants":2,"salaire":100000,"impôt":19884,"surcôte":4480,"décôte":0,"réduction":0,"taux":0.
    41}
5   {"marié":"oui","enfants":3,"salaire":100000,"impôt":9200,"surcôte":2180,"décôte":0,"réduction":0,"taux":0.3
    }
6   {"marié":"non","enfants":0,"salaire":100000,"impôt":22986,"surcôte":0,"décôte":0,"réduction":0,"taux":0.41}
7   {"marié":"oui","enfants":2,"salaire":30000,"impôt":0,"surcôte":0,"décôte":0,"réduction":0,"taux":0}
8   {"marié":"non","enfants":0,"salaire":200000,"impôt":64210,"surcôte":7498,"décôte":0,"réduction":0,"taux":0.
    45}
9   {"marié":"oui","enfants":3,"salaire":200000,"impôt":42842,"surcôte":17283,"décôte":0,"réduction":0,"taux":0
    .41}
```

et celui du fichier texte **[$errorsFileName]** le suivant :

```
1   la ligne 5 du fichier taxpayersdata.txt est erronée
2   la ligne 7 du fichier taxpayersdata.txt est erronée
```

1.8.5 La classe [Utilitaires]

Nous définissons par ailleurs une classe **[Utilitaires]** dans un fichier **[Utilitaires.php]** :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  // une classe de fonctions utilitaires
7.  abstract class Utilitaires {
8.
9.    public static function cutNewLinechar(string $ligne): string {
10.     // on supprime la marque de fin de ligne de $ligne si elle existe
11.     $longueur = strlen($ligne);  // longueur ligne
12.     while (substr($ligne, $longueur - 1, 1) == "\n" or substr($ligne, $longueur - 1, 1) == "\r") {
13.       $ligne = substr($ligne, 0, $longueur - 1);
14.       $longueur--;
15.     }
16.     // fin - on rend la ligne
17.     return($ligne);
18.   }
19. }
```

Commentaires

- ligne 4 : la classe **[Utilitaires]** est également placée dans l'espace de noms **[Application]** ;
- ligne 9 : la méthode **[cutNewLinechar]** enlève l'éventuel caractère de fin de ligne du texte qu'on lui a passé en paramètre. Elle rend la nouvelle ligne ainsi formée. On notera que c'est une méthode **statique**, c'est à dire qu'elle sera appelée sous la former **[Utilitaires::cutNewLineChar]** ;

1.8.6 La classe abstraite [AbstractBaseImpots]

L'interface **[InterfaceImpots]** sera implémentée par la classe abstraite **[AbstractBaseImpots]** suivante **[AbstractBaseImpots.php]** :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  // définition d'une classe abstraite AbstractBaseImpots
7.  abstract class AbstractBaseImpots implements InterfaceImpots {
8.    // les données de l'administration fiscale
9.    private $taxAdminData = NULL;
10.
11.   // données nécessaires au calcul de l'impôt
12.   abstract function getTaxAdminData(): TaxAdminData;
13.
14. // calcul de l'impôt
15. // ----------------------------------------------------------------------------
16.   public function calculerImpot(string $marié, int $enfants, int $salaire): array {
17.     // $marié : oui, non
18.     // $enfants : nombre d'enfants
19.     // $salaire : salaire annuel
20.     // $this->taxAdminData : données de l'administration fiscale
21.     //
22.     // on vérifie qu'on a bien les données de l'administration fiscale
23.     if ($this->taxAdminData === NULL) {
24.       $this->taxAdminData = $this->getTaxAdminData();
25.     }
26.     // calcul de l'impôt avec enfants
27.     $result1 = $this->calculerImpot2($marié, $enfants, $salaire);
28.     $impot1 = $result1["impôt"];
29.     // calcul de l'impôt sans les enfants
30.     if ($enfants != 0) {
31.       $result2 = $this->calculerImpot2($marié, 0, $salaire);
32.       $impot2 = $result2["impôt"];
33.       // application du plafonnement du quotient familial
34.       $plafonDemiPart = $this->taxAdminData->getPlafondQfDemiPart();
35.       if ($enfants < 3) {
36.         // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants
37.         $impot2 = $impot2 - $enfants * $plafonDemiPart;
38.       } else {
39.         // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants, le double pour les suivants
40.         $impot2 = $impot2 - 2 * $plafonDemiPart - ($enfants - 2) * 2 * $plafonDemiPart;
41.       }
42.     } else {
43.       $impot2 = $impot1;
44.       $result2 = $result1;
45.     }
46.     // on prend l'impôt le plus fort
47.     if ($impot1 > $impot2) {
48.       $impot = $impot1;
49.       $taux = $result1["taux"];
50.       $surcôte = $result1["surcôte"];
51.     } else {
52.       $surcôte = $impot2 - $impot1 + $result2["surcôte"];
53.       $impot = $impot2;
54.       $taux = $result2["taux"];
55.     }
56.     // calcul d'une éventuelle décôte
57.     $décôte = $this->getDecôte($marié, $salaire, $impot);
58.     $impot -= $décôte;
59.     // calcul d'une éventuelle réduction d'impôts
60.     $réduction = $this->getRéduction($marié, $salaire, $enfants, $impot);
61.     $impot -= $réduction;
62.     // résultat
63.     return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" => $réduction,
    "taux" => $taux];
64.   }
65.
66. // ----------------------------------------------------------------------------
67.   private function calculerImpot2(string $marié, int $enfants, float $salaire): array {
68.     …
69.     // résultat
70.     return ["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]];
71.   }
72.
73.   // revenuImposable=salaireAnnuel-abattement
74.   // l'abattement a un min et un max
75.   private function getRevenuImposable(float $salaire): float {
76.     …
77.     // résultat
78.     return floor($revenuImposable);
79.   }
80.
```

```
81.  // calcule une décôte éventuelle
82.    private function getDecôte(string $marié, float $salaire, float $impots): float {
83.      …
84.      // résultat
85.      return ceil($décôte);
86.    }
87.
88.  // calcule une réduction éventuelle
89.    private function getRéduction(string $marié, float $salaire, int $enfants, float $impots): float {
90.      …
91.      // résultat
92.      return ceil($réduction);
93.    }
94.
95.    public function executeBatchImpots(string $usersFileName, string $resultsFileName, string $errorsFileName): void {
96.      …
97.    }
98.
99.  }
```

Commentaires

- ligne 4 : la classe **[AbstractBaseImpots]** sera dans l'espace de noms **[Application]** comme les autres éléments de l'application en cours d'écriture ;
- ligne 7 : la classe **[AbstractBaseImpots]** implémente l'interface **[InterfaceImpots]** ;
- ligne 9 : les données de l'administration fiscale seront placées dans l'attribut **[$taxAdminData] ;**
- ligne 12 : implémentation de la méthode **[getTaxAdminData]** de l'interface. On ne sait pas encore définir cette méthode : nous avons vu un exemple où les données de l'administration fiscale ont été prises dans un fichier jSON au paragraphe. Nous verrons un autre cas où les données seront à chercher dans une base de données. Ce sera aux classes dérivées de définir le contenu de la méthode **[getTaxAdminData]**. Les deux cas précédents donneront naissance à deux classes dérivées. La méthode **[getTaxAdminData]** est donc déclarée abstraite ce qui automatiquement rend la classe elle-même abstraite (ligne 7) ;
- lignes 15-64 : la fonction de calcul de l'impôt déjà rencontrée aux paragraphes lien et lien ;
- la version 02 mettait les données de l'administration fiscale dans un tableau associatif **[$taxAdminData]**. La version 03 les met dans l'attribut **[$this→taxAdminData]**. La 1re différence entre ces deux solutions est une différence de visibilité des données fiscales :
  - dans la version 02, le tableau associatif **[$taxAdminData]** n'avait pas une visibilité globale. Il était donc passé en paramètre à toutes les fonctions de calcul de l'impôt ;
  - dans la version 03, l'attribut **[$this→taxAdminData]** a une visibilité globale pour toutes les méthodes de la classe. Il n'est donc pas passé en paramètre à toutes les fonctions de calcul de l'impôt ;
- une seconde différence vient du fait que la version 03 remplace des fonctions par des méthodes de classe. Chaque appel de méthode se fait désormais avec une expression **[$this→getMéthode(…)]** (lignes 27, 31, 57, 60) ;
- une troisième différence est que lorsque la méthode **[calculerImpot]** démarre son travail, elle ne sait pas si l'attribut **[private $taxAdminData]** dont elle a besoin a été initialisé. En effet, le constructeur de la classe ne l'initialise pas. C'est donc à la méthode **[calculerImpot]** de le faire à l'aide de la méthode **[getTaxAdminData]** de la ligne 12. C'est ce qui est fait aux lignes 23-25 ;
- en-dehors de ces différences, les méthodes de calcul de l'impôt restent ce qu'elles étaient dans les versions précédentes ;

La fonction **[executeBatchImpots]** est la suivante :

```
1.     public function executeBatchImpots(string $usersFileName, string $resultsFileName, string $errorsFileName): void {
2.       // pas mal d'erreurs peuvent se produire dès qu'on gère des fichiers
3.       try {
4.         // ouverture fichier des erreurs
5.         $errors = fopen($errorsFileName, "w");
6.         if (!$errors) {
7.           throw new ExceptionImpots("Impossible de créer le fichier des erreurs [$errorsFileName]", 10);
8.         }
9.         // ouverture fichier des résultats
10.        $results = fopen($resultsFileName, "w");
11.        if (!$results) {
12.          throw new ExceptionImpots("Impossible de créer le fichier des résultats [$resultsFileName]", 11);
13.        }
14.        // lecture des données utilisateur
15.        // chaque ligne a la forme statut marital, nombre d'enfants, salaire annuel
16.        $data = fopen($usersFileName, "r");
17.        if (!$data) {
18.          throw new ExceptionImpots("Impossible d'ouvrir en lecture les déclarations des contribuables [$usersFileName]",
    12);
19.        }
20.        // on exploite la ligne courante du fichier des données utilisateur
21.        // qui a la forme statut marital, nombre d'enfants, salaire annuel
22.        $num = 1;          // n° ligne courante
23.        $nbErreurs = 0;    // nbre d'erreurs rencontrées
24.        while ($ligne = fgets($data, 100)) {
25.          // debug
26.          //  print "ligne n° " . ($i + 1) . " : " . $ligne;
27.          // on enlève l'éventuelle marque de fin de ligne
28.          $ligne = Utilitaires::cutNewLineChar($ligne);
```

```
29.         // on récupère les 3 champs marié:enfants:salaire qui forment $ligne
30.         list($marié, $enfants, $salaire) = explode(",", $ligne);
31.         // on les vérifie
32.         // le statut marital doit être oui ou non
33.         $marié = trim(strtolower($marié));
34.         $erreur = ($marié !== "oui" and $marié !== "non");
35.         if (!$erreur) {
36.           // le nombre d'enfants doit être un entier
37.           $enfants = trim($enfants);
38.           if (!preg_match("/^\s*\d+\s*$/", $enfants)) {
39.             $erreur = TRUE;
40.           } else {
41.             $enfants = (int) $enfants;
42.           }
43.         }
44.         if (!$erreur) {
45.           // le salaire est un entier sans les centimes d'euros
46.           $salaire = trim($salaire);
47.           if (!preg_match("/^\s*\d+\s*$/", $salaire)) {
48.             $erreur = TRUE;
49.           } else {
50.             $salaire = (int) $salaire;
51.           }
52.         }
53.         // erreur ?
54.         if ($erreur) {
55.           fputs($errors, "la ligne [$num] du fichier [$usersFileName] est erronée\n");
56.           $nbErreurs++;
57.         } else {
58.           // on calcule l'impôt
59.           $result = $this->calculerImpot($marié, (int) $enfants, (int) $salaire);
60.           // on inscrit le résultat dans le fichier des résultats
61.           $result = ["marié" => $marié, "enfants" => $enfants, "salaire" => $salaire] + $result;
62.           fputs($results, \json_encode($result, JSON_UNESCAPED_UNICODE) . "\n");
63.         }
64.         // ligne suivante
65.         $num++;
66.       }
67.       // des erreurs ?
68.       if ($nbErreurs > 0) {
69.         throw new ExceptionImpots("Il y a eu des erreurs", 15);
70.       }
71.     } catch (ExceptionImpots $ex) {
72.       // on relance l'exception
73.       throw $ex;
74.     } finally {
75.       // on ferme tous les fichiers
76.       fclose($data);
77.       fclose($results);
78.       fclose($errors);
79.     }
80.   }
```

Commentaires du code

- ligne 1 : la fonction reçoit trois paramètres :
  - **[$usersFileName]** : le nom du fichier texte contenant les données des contribuables. Chaque ligne de texte contient les données d'un contribuable sous la forme : statut marital (oui / non), nombre d'enfants, salaire annuel :

```
oui,2,55555
oui,2,50000
```

  - **[$resultsFileName]** : le nom du fichier texte qui contiendra les résultats. Chaque ligne de texte aura la forme suivante :

```
{"marié":"oui","enfants":2,"salaire":50000,"impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.14}
{"marié":"oui","enfants":3,"salaire":50000,"impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14}
```

  - **[$errorsFileName]** : le nom du fichier texte des erreurs :

```
la ligne [5] du fichier [taxpayersdata.txt] est erronée
la ligne [7] du fichier [taxpayersdata.txt] est erronée
```

- ligne 3 : parce qu'un certain nombre d'opérations peuvent lancer une exception, un try / catch / finally entoure tout le code de la méthode ;
- lignes 3-19 : les trois fichiers sont ouverts. Une exception est lancée dès qu'une ouverture échoue ;
- ligne 24 : les lignes du fichier **[$data]** sont lues une par une à raison de 100 caractères au plus (les lignes font toutes moins de 100 caractères) ;
- ligne 28 : on utilise la méthode statique **[Utilitaires::cutNewLineChar]** pour enlever l'éventuelle marque de fin de ligne ;
- ligne 30 : on récupère les trois éléments de la ligne lue ;

- lignes 33-52 : la validité des trois éléments est vérifiée. Ici, on ne lance pas une exception s'il y a eu erreur mais on écrit le message de celle-ci dans le fichier texte **[$errors]** (ligne 55) ;
- ligne 59 : si la ligne lue est valide, le calcul de l'impôt est fait. On obtient un résultat sous forme de tableau associatif **["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" => $réduction, "taux" => $taux]** ;
- ligne 61 : au résultat obtenu, on ajoute les clés **[marié, enfants, salaire]** ;
- ligne 61 : le résultat est inscrit dans le fichier texte **[$results]** sous la forme de la chaîne jSON du résultat obtenu ;
- lignes 68-70 : à la fin de l'exploitation du fichier **[$data]**, on regarde le nombre de lignes erronées rencontrées. S'il y en a au moins une, on lance une exception ;
- lignes 71-74 : on intercepte l'exception qu'a pu lancer le code et on la relance immédiatement (ligne 73). Le but de cet artifice est de pouvoir avoir une clause **[finally]** aux lignes 74-79 : quelque soit la façon dont se termine l'exécution du code de la méthode, les trois fichiers qui ont pu être ouverts par ce code sont fermés. Fermer un fichier qui n'a pas été ouvert ne provoque pas d'erreur ;

1.8.7 La classe [ImpotsWithTaxAdminDataInJsonFile]

La classe abstraite **[AbstractBaseImpots]** n'implémente pas la méthode **[getTaxAdminData]** de l'interface **[InterfaceImpots]**. Il nous faut donc la définir dans une classe dérivée. Nous le faisons dans la classe dérivée **[ImpotsWithTaxAdminDataInJsonFile]** suivante :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  // définition d'une classe ImpotsWithDataInArrays
7.  class ImpotsWithTaxAdminDataInJsonFile extends AbstractBaseImpots {
8.    // un attribut de type Data
9.    private $taxAdminData;
10.
11.   // le constructeur
12.   public function __construct(string $jsonFileName) {
13.     // on initialise $this->taxAdminData à partir du fichier JSON
14.     $this->taxAdminData = (new TaxAdminData())->setFromJsonFile($jsonFileName);
15.   }
16.
17.   // retourne les données permettant le calcul de l'impôt
18.   public function getTaxAdminData(): TaxAdminData {
19.     // on rend l'attribut [$this->taxAdminData]
20.     return $this->taxAdminData;
21.   }
22.
23. }
```

Commentaires

- ligne 7 : la classe **[ImpotsWithTaxAdminDataInJsonFile]** étend la classe abstraite **[AbstractBaseImpots]**. Elle aura à définir la méthode **[getTaxAdminData]** que sa classe parent n'a pas définie ;
- ligne 9 : l'attribut **[$taxAdminData]** contiendra les données de l'administration fiscale ;
- lignes 12-15 : le constructeur reçoit comme unique paramètre le nom du fichiet jSON contenant les données fiscales ;
- ligne 14 : un objet de type **[TaxAdminData]** est créé puis initialisé. Cette opération peut lancer une exception de type **[ExceptionImpots]**. Celle-ci remontera jusqu'au script principal **[main.php]** ;
- lignes 18-20 : on donne un corps à la méthode **[getTaxAdminData]** que la classe parent n'avait pas définie. Ici, il suffit de rendre l'attribut **[$this->taxAdminData]** initialisé par le constructeur ;

1.8.8 Le script [main.php]

Ces classes et interface sont exploitées par le script **[main.php]** suivant :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare(strict_types = 1);
5.
6.  // espace de noms
7.  namespace Application;
8.
9.  // inclusion interface et classes
10. require_once __DIR__ . '/InterfaceImpots.php';
11. require_once __DIR__ . "/TaxAdminData.php";
12. require_once __DIR__ . '/ExceptionImpots.php';
13. require_once __DIR__ . '/Utilitaires.php';
14. require_once __DIR__ . '/AbstractBaseImpots.php';
```

```
15. require_once __DIR__ . "/ImpotsWithTaxAdminDataInJsonFile.php";
16.
17. // test ----------------------------------------------------
18. // définition des constantes
19. const TAXPAYERSDATA_FILENAME = "taxpayersdata.txt";
20. const RESULTS_FILENAME = "resultats.txt";
21. const ERRORS_FILENAME = "errors.txt";
22. const TAXADMINDATA_FILENAME = "taxadmindata.json";
23.
24. try {
25.   // on crée un objet ImpotsWithTaxAdminDataInJsonFile
26.   $impots = new ImpotsWithTaxAdminDataInJsonFile(TAXADMINDATA_FILENAME);
27.   // on exécute le batch des impôts
28.   $impots->executeBatchImpots(TAXPAYERSDATA_FILENAME, RESULTS_FILENAME, ERRORS_FILENAME);
29. } catch (ExceptionImpots $ex) {
30.   // on affiche l'erreur
31.   print $ex->getMessage() . "\n";
32. }
33. // fin
34. print "Terminé\n";
35. exit();
```

Commentaires

- ligne 4 : on impose le respect strict des types des paramètres des fonctions ;
- ligne 7 : le script **[main.php]** est lui aussi placé dans l'espace de noms **[Application]** ;
- lignes 10-15 : on indique à l'interpréteur PHP où se trouvent les classes et interfaces utilisées par le script. On notera qu'ici nous n'avons pas utilisé d'intruction *use* pour déclarer le nom complet des classes utilisées par le script. C'est en effet inutile parce que le script et les classes sont dans le même espace de noms **[Application]** ;
- lignes 18-22 : les noms des fichiers texte utilisés dans le script ;
- lignes 24-29 : un objet **[ImpotsWithTaxAdminDataInJsonFile]** est créé et l'éventuelle exception est gérée ;
- ligne 28 : on exécute la méthode **[executeBatchImpots]** qui va faire le calcul des impôts pour tous les contribuables du fichier **[TAXPAYERSDATA_FILENAME]**. Les résultats seront mis dans le fichier **[RESULTS_FILENAME]** et les erreurs éventuelles dans le fichier **[ERRORS_FILENAME]** ;
- lignes 29-32 : en cas d'erreur irrécupérable, on affiche le message de l'erreur ;

Résultats

Avec le fichier des contribuables **[taxpayersdata.txt]** suivants :

```
1   oui,2,55555
2   oui,2,50000
3   oui,3,50000
4   non,2,100000
5   non,3X,100000
6   oui,3,100000
7   oui,5,100000X
8   non,0,100000
9   oui,2,30000
10  non,0,200000
11  oui,3,200000
```

on obtient le fichier des erreurs **[errors.txt]** suivant :

```
1   la ligne [5] du fichier [taxpayersdata.txt] est erronée
2   la ligne [7] du fichier [taxpayersdata.txt] est erronée
```

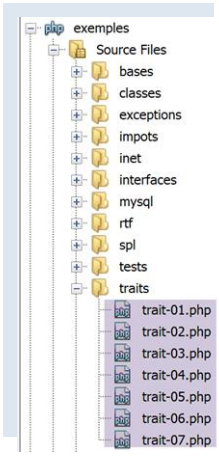
et le fichier des résultats **[resultats.txt]** suivant :

```
1   {"marié":"oui","enfants":2,"salaire":55555,"impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.14}
2   {"marié":"oui","enfants":2,"salaire":50000,"impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.14}
3   {"marié":"oui","enfants":3,"salaire":50000,"impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14}
4   {"marié":"non","enfants":2,"salaire":100000,"impôt":19884,"surcôte":4480,"décôte":0,"réduction":0,"taux":0.41}
5   {"marié":"oui","enfants":3,"salaire":100000,"impôt":9200,"surcôte":2180,"décôte":0,"réduction":0,"taux":0.3}
6   {"marié":"non","enfants":0,"salaire":100000,"impôt":22986,"surcôte":0,"décôte":0,"réduction":0,"taux":0.41}
7   {"marié":"oui","enfants":2,"salaire":30000,"impôt":0,"surcôte":0,"décôte":0,"réduction":0,"taux":0}
8   {"marié":"non","enfants":0,"salaire":200000,"impôt":64210,"surcôte":7498,"décôte":0,"réduction":0,"taux":0.45}
9   {"marié":"oui","enfants":3,"salaire":200000,"impôt":42842,"surcôte":17283,"décôte":0,"réduction":0,"taux":0.41}
```

1.9 Les Traits

Les **Traits** sont des structures analogues à des classes. Néanmoins on ne peut les instancier. Elles sont destinées à être incluses dans des classes. L'inclusion d'un **Trait** dans une classe a le même effet que si l'on avait copié le code du **Trait** dans la classe. On met dans un **Trait** du code susceptible d'être réutilisé dans plusieurs classes.

1.9.1 L'arborescence des scripts



1.9.2 Inclusion d'un trait dans une classe

Le script **[traits-01.php]** montre une utilisation basique d'un trait dans une classe :

```
1.  <?php
2.
3.  class Class04 {
4.    // attribut
5.    private $name;
6.
7.    // constructeur
8.    public function __construct(string $name) {
9.      $this->name = $name;
10.   }
11.
12.   // getters et setters
13.   public function getName(): string {
14.     return $this->name;
15.   }
16.
17.   public function setName(string $name): void {
18.     $this->name = $name;
19.   }
20.
21. }
22.
23. trait Trait04 {
24.   // attribut
25.   private $name;
26.
27.   // getters et setters
28.   public function getName(): string {
29.     return $this->name;
30.   }
31.
32.   public function setName(string $name): void {
33.     $this->name = $name;
34.   }
35.
36. }
37.
38. class Class05 {
39.   // inclusion du Trait
40.   use Trait04;
41.
42.   // constructeur
```

```
43.   public function __construct(string $name) {
44.     $this->name = $name;
45.   }
46.
47. }
48.
49. // test --------------
50. $class04 = new Class04("Tim");
51. $class05 = new Class05("Burton");
52. print $class04->getName() . "\n";
53. print $class05->getName() . "\n";
54. // affichage des deux classes
55. print_r($class04);
56. print_r($class05)
```

Commentaires du code

- lignes 3-21 : définition de la classe **[Class04]** avec un attribut, ses get / set et un constructeur ;
- lignes 23-36 : on reprend le code de **[Class04]** sans son constructeur et on le transfère dans le trait **[Trait04]** tel quel. On ne reprend pas le constructeur puisqu'un Trait n'est pas instanciable ;
- ligne 23 : c'est le mot clé **[trait]** qui fait de **[Trait04]** un trait plutôt qu'une classe ;
- lignes 38-45 : on définit une classe **[Class05]** qui reprend le code du trait **[Trait04]** (ligne 40) et lui ajoute un constructeur (lignes 43-45), identique à celui de la classe **[Class04]** pour rendre la classe instanciable ;
- ligne 40 : c'est le mot clé **[use]** qui permet l'inclusion d'un Trait dans une classe ;
- lignes 50-56 : des tests montrent que les classes **[Class04]** et **[Class05]** fonctionnent de la même façon ;

Résultats

```
1   Tim
2   Burton
3   Class04 Object
4   (
5       [name:Class04:private] => Tim
6   )
7   Class05 Object
8   (
9       [name:Class05:private] => Burton
10  )
```

Les résultats des lignes 3-10 montrent que les classes **[Class04]** et **[Class05]** ont la même contenu ;

Conclusion

L'utilisation de l'instruction **[use Trait]** dans une classe est équivalente à inclure le code de **[Trait]** dans la classe.

1.9.3 Utiliser un même trait dans différentes classes

Un premier intérêt du **trait** semble être la réutilisation d'un même code (attributs +méthodes) entre différentes classes. Nous allons voir cependant qu'on peut arriver au même objectif en utilisant de simples classes.

Le partage d'un *trait* entre classes est illustré par le script **[trait-02.php]** suivant :

```
1.  <?php
2.
3.  trait Trait01 {
4.    // attribut
5.    private $id = 0;
6.
7.    // méthode
8.    public function doSomething() {
9.      print "Trait01::doSomething... ($this->id)\n";
10.   }
11.
12. }
13.
14. class Class02 {
15.   // inclusion Trait01
16.   use Trait01 {
17.     // la méthode [Trait01::doSomething] est accessible
18.     // dans la classe sous le nom [doSomethingInTrait]
19.     Trait01::doSomething as doSomethingInTrait;
20.   }
```

```
21.
22.   // méthode propre à la classe
23.   public function doSomething(): void {
24.     // attribut id
25.     $this->id += 10;
26.     // utilisation méthode de Trait01
27.     $this->doSomethingInTrait();
28.     // afichage local
29.     print "Class02->doSomething\n";
30.   }
31.
32. }
33.
34. class Class03 {
35.   // inclusion Trait01
36.   use Trait01;
37.
38.   // méthode locale à la classe
39.   public function doSomethingElse(): void {
40.     // attribut id
41.     $this->id += 10;
42.     // utilisation méthode de Trait01
43.     $this->doSomething();
44.     // affichage local
45.     print "Class03->doSomethingElse\n";
46.   }
47.
48. }
49.
50. // test01 ----------------
51. function test01(): void {
52.   $class02 = new Class02();
53.   $class03 = new Class03();
54.   $class02->doSomething();
55.   $class03->doSomethingElse();
56. }
57.
58. // test01
59. print "test01-----------------\n";
60. test01();
```

Commentaires

- lignes 3-10 : un trait définissant un attibut (ligne 5) et une méthode (lignes 8-10).
- le trait **[Trait01]** est injecté dans deux classes **[Class02]** (lignes 14-32) et **[Class03]** (lignes 34-46).
- lignes 16-20 : injection de **[Trait01]** dans **[Class02]** ;
- la ligne 19 vise à résoudre un conflit : **[Trait01]** et **[Class02]** ont tous les deux une méthode appelée **[doSomething]**. Il y a deux cas à prévoir :
  - la méthode **[Class02::doSomething]** est appelée de l'extérieur de la classe. Dans ce cas, la méthode **[Class02::doSomething]** est prioritaire sur la méthode **[Trait01::doSomething]** et c'est elle qui est appelée ;
  - la méthode **[Class02::doSomething]** est appelée de l'intérieur de la classe. Dans ce cas il y a conflit : l'interpréteur PHP ne sait pas quelle méthode appeler ;

  La ligne 19 permet de renommer **[doSomethingInTrait]** la méthode **[Trait01::doSomething]**. Ainsi, à l'intérieur de **[Class02]** on utilisera les notations :
  - **[doSomethingInTrait]** pour appeler la méthode **[Trait01::doSomething]** ;
  - **[doSomething]** pour appeler la méthode **[Class02::doSomething]** ;
- lignes 25, 27 : la classe **[Class02]** utilisent l'attribut et la méthode de **[Trait01]** comme s'ils lui étaient propres ;
- lignes 34-48 : la classe **[Class03]** est identique à la classe **[Class02]**. L'inclusion de **[Trait01]** est ici plus simple parce qu'il n'y pas collision entre les méthodes de **[Trait01]** et **[Class03]** ;

Résultats

```
1   test01-----------------
2   Trait01::doSomething… (10)
3   Class02->doSomething
4   Trait01::doSomething… (10)
5   Class03->doSomethingElse
```

On notera bien qu'il n'y a pas partage du trait **[Trait01]** entre les classes **[Class02]** et **[Class03]**. Ainsi l'attribut **[Trait01::i]** devient par inclusion de **[Trait01]** dans les classes **[Class02]** et **[Class03]** deux attributs différents **[Class02::i]** et **[Class03::i]**. C'est ce que montrent les lignes 2 et 4 des résultats. Si l'attribut **[Trait01::i]** avait été partagé entre les classes **[Class02]** et **[Class03]** on aurait eu 20 à la ligne 4 au lieu de 10.

Le script **[trait-03.php]** montre qu'on peut arriver au même résultat en utilisant une classe au lieu du trait :

```php
<?php

// classe qui remplace le trait
class Class01 {
  // attribut
  private $id = 0;

  // setter
  public function setId(int $id) {
    $this->id = $id;
  }

  // getter
  public function getId(): int {
    return $this->id;
  }

  // méthode
  public function doSomething(): void {
    print "Class01::doSomething... ($this->id)\n";
  }

}
class Class02 {
  // inclusion Class01
  private $class01;

  // setter
  public function setClass01(Class01 $class01) {
    $this->class01 = $class01;
  }

  // méthode propre à la classe
  public function doSomething(): void {
    // chgt attribut de Class01
    $id = $this->class01->getId();
    $id += 10;
    $this->class01->setId($id);
    // utilisation méthode de Class01
    $this->class01->doSomething();
    // afichage local
    print "Class02->doSomething\n";
  }

}

class Class03 {
  // inclusion Class01
  private $class01;

  // setter
  public function setClass01(Class01 $class01) {
    $this->class01 = $class01;
  }

  // méthode locale à la classe
  public function doSomethingElse(): void {
    // chgt attribut de Class01
    $id = $this->class01->getId();
    $id += 10;
    $this->class01->setId($id);
    // utilisation méthode de Class01
    $this->class01->doSomething();
    // affichage local
    print "Class03->doSomethingElse\n";
  }

}

// test01 ----------------
function test01(): void {
  // deux objets
  $class02 = new Class02();
```

```
75.   $class03 = new Class03();
76.   // vont partager deux instances différentes de [Class01]
77.   $class02->setClass01(new Class01());
78.   $class03->setClass01(new Class01());
79.   // vérification
80.   $class02->doSomething();
81.   $class03->doSomethingElse();
82. }
83.
84. // test02 ----------------
85. function test02(): void {
86.   // instance partagée de [Class01]
87.   $class01 = new Class01();
88.   // deux objets
89.   $class02 = new Class02();
90.   $class03 = new Class03();
91.   // vont partager la même instance de [Class01]
92.   $class02->setClass01($class01);
93.   $class03->setClass01($class01);
94.   // vérification
95.   $class02->doSomething();
96.   $class03->doSomethingElse();
97. }
98.
99. // test01
100.print "test01-----------------\n";
101.test01();
102.// test02
103.print "test02-----------------\n";
104.test02();
```

Commentaires

- lignes 4-23 : la classe **[Class01]** remplace le trait **[Trait01]**. Le code de **[Trait01]** était incorporé au code des classes **[Class02]** et **[Class03]**. Ici, ce ne sera pas le cas. Ce sera une référence au code de la classe **[Class01]** qui sera injectée dans les classes **[Class02]** et **[Class03]**. Ce qui fait que les attributs de la classe **[Class01]** seront extérieurs au code des **[Class02]** et **[Class03]**. Comme ici, l'attribut **[id]** est privé (ligne 6), il faut prévoir un getter (lignes 14-16) et un setter (lignes 9-11). C'est la 1re différence avec le trait : il faut créer le code d'accès aux attributs privés de la classe. On aurait pu passer la visibilité de l'attribut **[id]** à **[public]** mais il n'est jamais conseillé de faire cela. Passer par un setter pour fixer la valeur d'un attribut permet d'en vérifier la validité ;
- ligne 27 : inclusion dans le code de **[Class02]** d'une référence à la classe **[Class01]**. Parce que cet attribut est privé, il nous faut créer un setter (lignes 30-32) pour l'initialiser ;
- lignes 37-41 : pour tout usage du code de **[Class01]**, il nous faut passer par l'attribut **[$this→class01]** ;
- lignes 48-69 : la classe **[Class03]** est un clône de la classe **[Classe02]** si ce n'est que sa méthode a un nom différent ;
- lignes 72-82 : le 1er test. Celui-ci consiste à injecter dans les classes **[Class02]** et **[Class03]** **deux instances différentes** de la classe **[Class01]** (lignes 77 et 78) ;
- lignes 85-97 : le 2e test injecte dans les classes **[Class02]** et **[Class03]** **la même instance** de la classe **[Class01]** (lignes 97, 92, 93) ;
- lignes 100-101 : exécution du test **[test01]** ;
- lignes 103-104 : exécution du test **[test02]** ;

Résultats

```
1   test01-----------------
2   Class01::doSomething… (10)
3   Class02->doSomething
4   Class01::doSomething… (10)
5   Class03->doSomethingElse
6   test02-----------------
7   Class01::doSomething… (10)
8   Class02->doSomething
9   Class01::doSomething… (20)
10  Class03->doSomethingElse
```

Commentaires des résultats

- lignes 2-5 : on obtient les mêmes résultats qu'avec le trait **[Trait01]**. On en conclura que l'usage du trait n'est ici pas indispensable mais qu'il amène une réduction du code dû au fait qu'il n'y a pas besoin de méthodes pour accéder aux attributs du trait : ceux-ci font partie intégrante du code dans lequel le trait a été incorporé ;
- lignes 7-10 : du fait qu'on a injecté la même référence de **[Class01]** dans les classes **[Class02]** et **[Class03]**, l'attribut **[Class01::id]** a été partagé entre les deux classes. C'est pour cela que la ligne 9 des résultats affiche 20 al lieu de 10 avec le

trait. On en conclura que si des attributs du trait doivent être **partagés** entre des classes, alors le trait n'est pas utilisable et il faut alors utiliser une classe ;

1.9.4 Regrouper des méthodes dans un trait

Dans l'exemple précédent, le **trait** comportait attributs et méthodes. Nous considérons ici le cas où il ne contient que des méthodes. Dans ce cas, le **trait** ressemble à une factorisation de méthodes qu'on peut alors utiliser dans différentes classes. Comme le **trait** n'a ici pas d'attribut, on va considérer le cas où les méthodes qu'ils rassemblent travaillent uniquement sur des paramètres qu'on leur passe. En fait, ce n'est pas obligatoire : un trait peut travailler sur un attribut **[$this→attribut1]** sans avoir cet attribut. C'est alors aux classes qui utilisent ce trait de fournir l'attribut **[$this→attribut1]**.
Dans le cas où le **trait** n'a que des méthodes qui travaillent uniquement sur des paramètres qu'on leur passe, nous montrerons que le **trait** peut alors être remplacé par une **classe** ayant les mêmes méthodes que le **trait** et déclarées statiques.

L'utilisation du trait est illustré par le script **[trait-04.php]** qui reprend le **trait** de l'exemple précédent en lui enlevant tout attribut :

```
1.  <?php
2.
3.  trait Trait01 {
4.
5.    // méthode à partager
6.    public function doSomething() {
7.      print "Trait01::doSomething ....\n";
8.    }
9.
10. }
11.
12. class Class02 {
13.   // inclusion Trait01
14.   use Trait01 {
15.     // la méthode [Trait01::doSomething] est accessible
16.     // dans la classe sous le nom [doSomethingInTrait]
17.     Trait01::doSomething as doSomethingInTrait;
18.   }
19.
20.   public function doSomething(): void {
21.     // appel méthode de Trait01
22.     $this->doSomethingInTrait();
23.     // affichage local
24.     print "Class02->doSomething\n";
25.   }
26.
27. }
28.
29. class Class03 {
30.   // inclusion Trait01
31.   use Trait01;
32.
33.   // méthode locale à la classe
34.   public function doSomethingElse(): void {
35.     // appel méthode de Trait01
36.     $this->doSomething();
37.     // affichage local
38.     print "Class03->doSomethingElse\n";
39.   }
40.
41. }
42.
43. // test ----------------
44. (new Class02())->doSomething();
45. (new Class03())->doSomethingElse();
```

Commentaires

- lignes 3-10 : le trait **[Trait01]** n'a plus d'attributs ;
- lignes 14-18 : inclusion de **[Trait01]** dans **[Class02]**. La méthode de **[Trait01]** est utilisée ligne 22 ;
- ligne 31 : inclusion de **[Trait01]** dans **[Class03]**. La méthode de **[Trait01]** est utilisée ligne 36 ;

Résultats

```
1   Trait01::doSomething ….
2   Class02->doSomething
3   Trait01::doSomething ….
4   Class03->doSomethingElse
```

Dans ce cas d'usage, le trait **[Trait01]** peut aisément être remplacé par une classe. Ceci est montré par le script **[trait-05.php]** suivant :

```
1.  <?php
2.
3.  abstract class Class01 {
4.
5.    // méthode statique à partager
6.    public static function doSomething() {
7.      print "Class01::doSomething ....\n";
8.    }
9.
10. }
11.
12. class Class02 {
13.
14.   public function doSomething(): void {
15.     // appel méthode de Class01
16.     Class01::doSomething();
17.     // affichage local
18.     print "Class02->doSomething\n";
19.   }
20.
21. }
22.
23. class Class03 {
24.
25.   // méthode locale à la classe
26.   public function doSomethingElse(): void {
27.     // appel méthode de Class01
28.     Class01::doSomething();
29.     // affichage local
30.     print "Class03->doSomethingElse\n";
31.   }
32.
33. }
34.
35. // test ----------------
36. (new Class02())->doSomething();
37. (new Class03())->doSomethingElse();
```

Commentaires

- lignes 3-10 : le trait **[Trait01]** est remplacé par une classe abstraite **[Class01]** dont toutes les méthodes sont déclarées statiques. La classe est déclarée abstraite uniquement pour empêcher son **instanciation**. On aurait voulu écrire également **[final]** pour empêcher sa **dérivation** mais PHP 7 n'accepte pas le préfixe **[final abstract]** pour une classe. C'est l'un ou l'autre mais pas les deux ;
- ligne 16 : au lieu d'écrire **[$this→doSomethingInTrait]** on écrit maintenant **[Class01::doSomething]**, ç-à-d qu'on appelle la méthode statique **[doSomething]** de la classe **[Class01]** ;
- ligne 28 : on répète la même démarche dans **[Class03]** ;

Résultats

```
1   Class01::doSomething ….
2   Class02->doSomething
3   Class01::doSomething ….
4   Class03->doSomethingElse
```

On a bien le même résultat qu'avec le trait **[Trait01]** ce qui montre que l'usage de celui-ci peut être évité. On a écrit que les méthodes d'un trait peuvent travailler sur un attribut **[$this→attribut1]** sans que le trait ait cet attribut. C'est alors aux classes qui utilisent ce trait de fournir l'attribut **[$this→attribut1]**. C'est un cas exotique : autant 'remonter' l'attribut **[$this→attribut1]** que les classes utilisant le trait doivent avoir, dans le trait lui-même. Ainsi il fera forcément partie des attributs de la classe utilisant le trait.

1.9.5 Héritage multiple avec un trait

Il est fréquent de lire dans la littérature PHP que le trait permettrait l'héritage multiple : la possibilité pour une classe d'hériter de plusieurs classes. Le langage C++ possède cette possibilité mais pas les langages Java ou C# qui ne connaissent que l'héritage simple. Nous allons montrer que si effectivement l'usage d'un trait dans une classe dérivée permet d'implémenter quelque chose qui ressemble à l'héritage multiple, ce cas d'usage peut là encore s'implémenter avec de simples classes.

Le script **[trait-06.php]** met en œuvre une classe dérivée et un trait :

```
1.  <?php
2.
3.  trait Trait01 {
4.    // attribut
5.    private $i;
6.
7.    // méthode à partager
8.    public function doSomethingInTrait01() {
9.      // modification Trait01::$i
10.     $this->i++;
11.     // affichage
12.     print "Trait01::doSomethingInTrait01... i=$this->i\n";
13.   }
14.
15. }
16.
17. class Class02 {
18.   // attribut
19.   protected $j = 0;
20.
21.   // méthode
22.   public function doSomethingInClass02(): void {
23.     // modification Class02::j
24.     $this->j += 10;
25.     // affichage
26.     print "Class02->doSomethingInClass02... j=$this->j\n";
27.   }
28.
29. }
30.
31. // classe dérivée
32. class Class03 extends Class02 {
33.   // hérite de Class02:j et Trait01::i
34.   // inclusion Trait01
35.   use Trait01;
36.
37.   // méthode
38.   public function doSomethingInClass03(): void {
39.     // utilisation méthode de Trait01
40.     $this->doSomethingInTrait01();
41.     // modification Trait01::i
42.     $this->i += 100;
43.     // modification Class03::j (==Class02::j)
44.     $this->j += 1000;
45.     // affichage
46.     print "Class03->doSomethingInClass03... i=$this->i, j=$this->j\n";
47.   }
48.
49. }
50.
51. // test ----------------
52. (new Class02())->doSomethingInClass02();
53. (new Class03())->doSomethingInClass03();
```

Commentaires

- lignes 3-15 : on revient à un trait **[Trait01]** avec un attribut et une méthode manipulant celui-ci ;
- lignes 17-29 : une classe **[Class02]** qui n'a rien à voir avec le trait **[Trait01]**. Elle ne l'utilise pas ;
- ligne 19 : on a déclaré l'unique attribut de **[Class02]** avec une visibilité **[protected]** pour que celui-ci soit accessible dans les classes dérivées ;
- ligne 32 : la classe **[Class03]** étend la classe **[Class02]**. De plus elle incorpore le trait **[Trait01]** (ligne 35). Finalement, elle hérite des attributs et méthodes de **[Class02]** et incorpore les attributs et méthodes de **[Trait01]**. On a donc bien quelque chose d'analogue à l'héritage multiple ;

Résultats

```
1   Class02->doSomethingInClass02… j=10
2   Trait01::doSomethingInTrait01… i=1
3   Class03->doSomethingInClass03… i=101, j=1000
```

De la même façon qu'il a été fait dans un exemple précédent, nous allons montrer que :

- le trait peut être remplacé par une classe ;
- au lieu d'incorporer le trait dans la classe dérivée, on incorpore la référence d'une instance de la classe ;

Le script **[trait-07.php]** est le suivant :

```
1.  <?php
2.
3.  class Class01 {
4.    // attribut
5.    protected $i;
6.
7.    // getter et setter
8.    public function getI(): int {
9.      return $this->i;
10.   }
11.
12.   public function setI(int $i): void {
13.     $this->i = $i;
14.   }
15.
16.   // méthode
17.   public function doSomethingInClass01(): void {
18.     // modification Class01::$i
19.     $this->i++;
20.     // affichage
21.     print "Class01::doSomething in Class01... i=$this->i\n";
22.   }
23.
24. }
25.
26. class Class02 {
27.   // attribut
28.   protected $j = 0;
29.
30.   // méthode propre à la classe
31.   public function doSomethingInClass02(): void {
32.     // modification Class02::j
33.     $this->j += 10;
34.     // affichage
35.     print "Class02->doSomethingInClass02... j=$this->j\n";
36.   }
37.
38. }
39.
40. class Class03 extends Class02 {
41.   // inclusion Class01
42.   private $class01;
43.
44.   // setter
45.   public function setClass01(Class01 $class01) {
46.     $this->class01 = $class01;
47.   }
48.
49.   // méthode locale à la classe
50.   public function doSomethingInClass03(): void {
51.     // utilisation méthode de Class01
52.     $this->class01->doSomethingInClass01();
53.     // modification Class01::i
54.     $i = $this->class01->getI();
55.     $i += 100;
56.     $this->class01->setI($i);
57.     // modification Class03::j
58.     $this->j += 1000;
59.     // affichage
60.     print "Class03->doSomethingInClass03... i=$i, j=$this->j\n";
61.   }
62.
63. }
64.
65. // test ----------------
66. $class01 = new Class01();
67. $class02 = new Class02();
68. $class03 = new Class03();
69. $class03->setClass01($class01);
70. $class02->doSomethingInClass02();
71. $class03->doSomethingInClass03();
```

Commentaires

- lignes 3-24 : la classe **[Class01]** remplace le trait **[Trait01]**. Comme la classe **[Class01]** va être incorporée dans les classes via une référence, on a prévu des get / set pour l'attribut **[$i]** ;
- lignes 26-38 : la classe **[Classe02]** ne change pas ;
- ligne 40 : la classe **[Class03]** étend la classe **[Classe02]** ;
- ligne 42 : la classe **[Classe01]** est incorporée à **[Classe03]** via une référence ;
- lignes 45-47 : on prévoit un *setter* pour initialiser la référence à la classe **[Classe01]** ;
- lignes 50-61 : la méthode **[doSomethingInClass03]** fait la même chose que précédemment avec cependant un code plus complexe ;

Résultats

```
1   Class02->doSomethingInClass02… j=10
2   Class01::doSomething in Class01… i=1
3   Class03->doSomethingInClass03… i=101, j=1000
```

De cet exemple, on peut conclure que là encore le trait n'est pas indispensable mais il faut reconnaître qu'il permet l'écriture d'un code plus court dans la classe dérivée.

1.9.6 Utiliser un trait à la place d'une classe abstraite

On rencontre souvent le cas d'utilisation suivant : on crée une interface **I** assez générale qui peut donner naissance à plusieurs implémentations. Celles-ci partagent un code commun mais diffèrent par d'autres méthodes. On peut implémenter ce cas d'utilisation de deux façons :

1. on crée une classe **abstraite C** qui regroupe le code commun aux classes dérivées. La classe C implémente l'interface I mais certaines méthodes qui doivent être déclarées dans les classes dérivées sont dans la classe C déclarées abstraites et donc la classe C est elle-même abstraite. On crée ensuite des classes C1 et C2 dérivées de C qui implémentent chacune à leur manière les méthodes non définies (abstraites) de leur classe parent C ;
2. on crée un **trait T** quasi identique à la classe abstraite C de la solution précédente. Ce trait n'implémente pas l'interface I car syntaxiquement elle ne le peut pas. On crée ensuite des classes C1 et C2 implémentant l'interface I et utilisant le trait T. Il ne reste plus à ces classes qu'à implémenter les méthodes de l'interface I non implémentées par le trait T qu'elles importent ;

Voici un exemple qui montre la grande proximité de ces deux solutions.

L'application 1 implémente la solution 1 décrite précédemment **[trait-08.php]** :

```
1.  <?php
2.
3.  interface Interface1 {
4.
5.    public function doSomething(): void;
6.
7.    public function doSomethingElse(): void;
8.  }
9.
10. abstract class AbstractClass implements Interface1 {
11.   // attributs
12.   protected $attr1 = 11;
13.   protected $attr2 = 12;
14.
15.   // getters et setters
16.   public function getAttr1() {
17.     return $this->attr1;
18.   }
19.
20.   public function getAttr2() {
21.     return $this->attr2;
22.   }
23.
24.   public function setAttr1($attr1) {
25.     $this->attr1 = $attr1;
26.     return $this;
27.   }
28.
29.   public function setAttr2($attr2) {
30.     $this->attr2 = $attr2;
31.     return $this;
32.   }
```

```
33.
34.   // méthode implémentée
35.   public function doSomething(): void {
36.     print "AbstractClass::doSomething [$this->attr1,$this->attr2]\n";
37.   }
38.
39.   // méthode non implémentée
40.   abstract public function doSomethingElse(): void;
41. }
42.
43. // classe dérivée 1
44. class Class1 extends AbstractClass {
45.   // attribut
46.   private $attr3 = 13;
47.
48.   // getter et setter
49.   public function getAttr3() {
50.     return $this->attr3;
51.   }
52.
53.   public function setAttr3($attr3) {
54.     $this->attr3 = $attr3;
55.     return $this;
56.   }
57.
58.   // implémentation doSomethingElse
59.   public function doSomethingElse(): void {
60.     print "Class1::doSomethingElse [$this->attr1,$this->attr2,$this->attr3]\n";
61.   }
62.
63. }
64.
65. // classe dérivée 2
66. class Class2 extends AbstractClass {
67.   // attribut
68.   private $attr4 = 14;
69.
70.   public function getAttr4() {
71.     return $this->attr4;
72.   }
73.
74.   public function setAttr4($attr4) {
75.     $this->attr4 = $attr4;
76.     return $this;
77.   }
78.
79.   // implémentation doSomethingElse
80.   public function doSomethingElse(): void {
81.     print "Class2::doSomethingElse [$this->attr1,$this->attr2,$this->attr4]\n";
82.   }
83.
84. }
85.
86. // fonction externe
87. function useInterfaceWith(Interface1 $interface):void{
88.   $interface->doSomething();
89.   $interface->doSomethingElse();
90. }
91.
92. // tests
93. useInterfaceWith(new Class1());
94. useInterfaceWith(new Class2());
```

Commentaires

- lignes 3-8 : **l'interface `[Interface1]`** a deux méthodes ;
- lignes 10-41 : la **classe abstraite `[AbstractClass]`** implémente l'interface **`[Interface1]`** (ligne 10). Elle a deux attributs avec leurs getters et setters (lignes 12-32), implémente la méthode **`[doSomething]`** de l'interface **`[Interface1]`** (lignes 35-37) mais ne sait pas implémenter la méthode **`[doSomethingElse]`**. Celle-ci est donc déclarée abstraite (ligne 40). La classe abstraite **`[AbstractClass]`** ne peut être instanciée et pour servir à quelque chose elle doit obligatoirement être dérivée ;
- lignes 44-63 : la classe **Class1** étend la classe abstraite **`[AbstractClass]`** et donc implémente l'interface **`[Interface1]`** (ligne 14). Elle donne un corps à la méthode **`[doSomethingElse]`** que sa classe parent n'avait pas définie (lignes 59-61). Elle ajoute également un attribut à ceux de sa classe parent (lignes 46-56) ;

- lignes 66-82 : la classe **Class2** étend la classe abstraite **[AbstractClass]** et donc implémente l'interface **[Interface1]** (ligne 66). Elle donne un corps à la méthode **[doSomethingElse]** que sa classe parent n'avait pas définie (lignes 80-82). Elle ajoute également un attribut à ceux de sa classe parent (lignes 68-77) ;
- lignes 87-90 : la fonction **[useInterfaceWith]** reçoit en paramètre un type **[Interface1]** et appelle les deux méthodes de cette interface ;
- lignes 93-94 : on appelle la fonction **[useInterfaceWith]** la 1re fois avec un type **[Class1]** et la seconde fois avec un type **[Class2]**. C'est correct puisque ces deux types implémentent l'interface **[Interface1]** ;

Résultats

```
1   AbstractClass::doSomething [11,12]
2   Class1::doSomethingElse [11,12,13]
3   AbstractClass::doSomething [11,12]
4   Class2::doSomethingElse [11,12,14]
```

Maintenant nous implémentons la solution 2 avec le script **[trait-09.php]**. Cela consiste à remplacer la classe abstraite par un trait :

```
1.  <?php
2.
3.  interface Interface1 {
4.
5.    public function doSomething(): void;
6.
7.    public function doSomethingElse(): void;
8.  }
9.
10. trait Trait1 {
11.   // attributs
12.   private $attr1 = 11;
13.   private $attr2 = 12;
14.
15.   // getters et setters
16.   public function getAttr1() {
17.     return $this->attr1;
18.   }
19.
20.   public function getAttr2() {
21.     return $this->attr2;
22.   }
23.
24.   public function setAttr1($attr1) {
25.     $this->attr1 = $attr1;
26.     return $this;
27.   }
28.
29.   public function setAttr2($attr2) {
30.     $this->attr2 = $attr2;
31.     return $this;
32.   }
33.
34.   // méthode implémentée
35.   public function doSomething(): void {
36.     print "Trait::doSomething [$this->attr1,$this->attr2]\n";
37.   }
38. }
39.
40. // classe dérivée 1
41. class Class1 implements Interface1 {
42.   // utilisation du trait
43.   use Trait1;
44.   // attribut
45.   private $attr3 = 13;
46.
47.   // getter et setter
48.   public function getAttr3() {
49.     return $this->attr3;
50.   }
51.
52.   public function setAttr3($attr3) {
53.     $this->attr3 = $attr3;
54.     return $this;
55.   }
56.
57.   // implémentation doSomethingElse
58.   public function doSomethingElse(): void {
```

```
59.     print "Class1::doSomethingElse [$this->attr1,$this->attr2,$this->attr3]\n";
60.   }
61.
62. }
63.
64. // classe dérivée 2
65. class Class2 implements Interface1 {
66.   // utilisation du trait
67.   use Trait1;
68.   // attribut
69.   private $attr4 = 14;
70.
71.   public function getAttr4() {
72.     return $this->attr4;
73.   }
74.
75.   public function setAttr4($attr4) {
76.     $this->attr4 = $attr4;
77.     return $this;
78.   }
79.
80.   // implémentation doSomethingElse
81.   public function doSomethingElse(): void {
82.     print "Class2::doSomethingElse [$this->attr1,$this->attr2,$this->attr4]\n";
83.   }
84.
85. }
86.
87. // fonction externe utilisant l'interface
88. function useInterfaceWith(Interface1 $interface): void {
89.   $interface->doSomething();
90.   $interface->doSomethingElse();
91. }
92.
93. // tests
94. useInterfaceWith(new Class1());
95. useInterfaceWith(new Class2());
```

Commentaires

- lignes 3-8 : l'interface **[Interface1]** n'a pas changé ;
- lignes 10-41 : le trait **[Trait1]** remplace la classe abstraite **[AbstractClass]** de la solution 1. Le code est le même aux détails près suivants :
  - ligne 10 : le trait **[Trait1]** n'implémente pas l'interface **[Interface1]**. C'est syntaxiquement impossible ;
  - lignes 12-13 : l'attribut de visibilité **[protected]** des attributs de la classe abstraite **[AbstractClass]** devient ici **[private]**. Ces deux attributs visent à donner aux classes dérivées un accès direct aux attributs de la classe parent (protected) ou du trait (private) sans avoir à passer par les getters et setters ;
  - le trait **[Trait1]** ne déclare pas la méthode abstraite **[doSomethingElse]** ;
- lignes 41-62 : la classe **[Class1]** de la solution 2 est identique à la classe **[Class1]** de la solution 1 aux détails près suivants :
  - ligne 41 : la classe **[Class1]** implémente l'interface **[Interface1]** alors que dans la solution 1, elle étendait la classe abstraite **[AbstractClass]** ;
  - ligne 43 : elle utilise le trait **[Trait1]** pour implémenter une partie de l'interface ;
- lignes 65-85 : on peut faire les mêmes commentaires que pour **[Class1]** ;
- lignes 87-95 : le reste du code ne change pas ;

Résultats

```
1   Trait::doSomething [11,12]
2   Class1::doSomethingElse [11,12,13]
3   Trait::doSomething [11,12]
4   Class2::doSomethingElse [11,12,14]
```

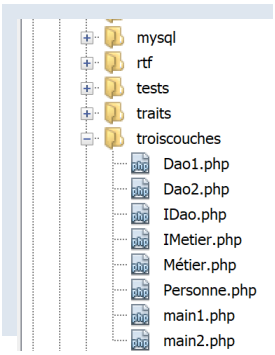
On obtient bien les mêmes résultats.

1.9.7 Conclusion

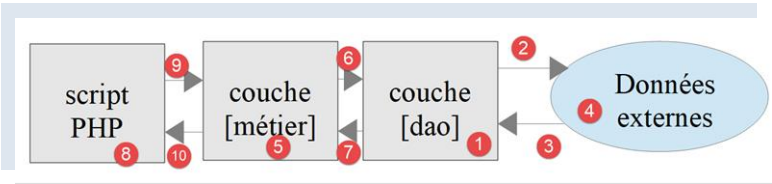
Des exemples précédents, il ressort que les cas d'utilisation où l'usage du trait amènerait un avantage net ne sont pas clairs. Sur nos exemples on peut toujours s'en passer en le remplaçant par une classe. Il semble cependant que son utilisation soit pratique pour factoriser du code entre différentes classes dérivées, comme si ce code appartenait à une classe parent. C'est ce que nous ferons dans un exemple à suivre.

1.10 Applications en couches

1.10.1 Introduction



Nous allons découvrir maintenant comment structurer une application PHP en couches :



Dans ce schéma, c'est la **couche de gauche qui prend l'initiative d'utiliser la couche de droite**. Le rôle des couches est le suivant :

- **[1]** : la couche appelée **[dao]** (Data Access Objects) s'occupe des échanges avec des entrepôts de données externes **[4]** (fichiers, bases de données, services web…). Cette couche est parfois appelée **[DAL]** (Data Access Layer) un terme qui décrit mieux le rôle de la couche. Cette couche peut lire des données **[3]** ou en écrire **[2]**. Elle est sollicitée par la couche **[métier]** **[6]** et lui rend des résultats **[7]** ;
- **[5]** :une couche appelée **[métier]** qui rassemble des procédures 'métier' à qui on fournit toutes les données dont elles ont besoin. C'est généralement la couche la plus stable d'un projet car elle ne dépend pas de la façon dont les données sont acquises. Celles-ci ont deux sources :
  - **[9]** : les données fournies par le script PHP ;
  - **[6,7]** : les données demandées à la couche **[dao]**.
- **[8]** : le script principal est le chef d'orchestre. Dans une application console, il va :
  - créer les couches **[métier]** et **[dao]** ;
  - dérouler l'algorithme de l'application. Cet algorithme est celui d'un chef d'orchestre : on n'y trouve aucun code 'métier' ou d'accès aux données. Le script principal se contente d'appeler les procédures de la couche **[métier]** **[9]**. Il ignore totalement la couche **[dao]** et les données externes. Il peut fournir à la couche **[métier]** des données **[9]**. Dans une application console, celles-ci peuvent provenir de fichiers de configuration ou de l'utilisateur du script. Il reçoit des résultats **[10]** de la couche **[métier]**. Il peut avoir besoin de stocker certains résultats : pour cela il utilise de nouveau les procédures de la couche **[métier]** qui elles vont s'adresser à la couche **[dao]** qui va faire le travail ;
  - parmi les résultats, le script principal peut recevoir des exceptions. C'est son rôle de gérer les exceptions qui remontent de toutes les couches ;

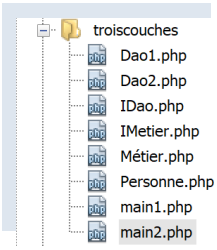
Ce découpage en couches est destiné à faciliter l'évolution de l'application. Pour cela, chaque couche implémente une interface. Supposons que :

- la couche **[dao]** est implémentée par une classe **[Dao]** qui implémente une interface **[IDao]** ;
- la couche **[métier]** est implémentée par une classe **[Métier]** qui implémente une interface **[IMétier]** ;

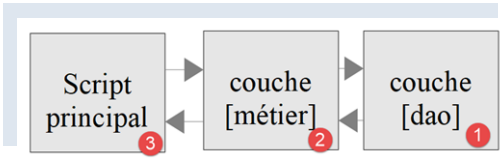
Les procédures de la couche **[métier]** vont alors utiliser l'interface **[IDao]** plutôt que la classe **[Dao]**. Ceci permet de faire évoluer la couche **[Dao]** sans toucher à la couche **[Métier]**. Supposons que dans une version 1, la couche **[Dao1]** utilise des données d'une base de données. Suite à une évolution, ces données sont maintenant fournies par un service web dans une version 2, **[Dao2]**. On s'assurera que les deux classes **[Dao1]** et **[Dao2]** implémentent la même interface **[IDao]** et qu'elles lancent les mêmes exceptions. Si ceci est vérifié, la couche **[Métier]** qui travaille avec l'interface **[IDao]** restera inchangée ;

Le même raisonnement s'applique à la couche **[Métier]**.

Voyons une implémentation avec des classes et interfaces :



L'application va implémenter la structure en couches suivante :



1.10.2 Objets échangés entre couches

En temps normal, les couches échangent divers objets. Ici elles échangeront la classe suivante **[Personne.php]** :

```
1.  <?php
2.
3.  // une personne
4.  class Personne {
5.    // identifiant
6.    private $id;
7.
8.    // constructeur
9.    public function __construct(int $id) {
10.     $this->id = $id;
11.   }
12.
13.   // toString
14.   public function __toString(): string {
15.     return "[Personne($this->id)]";
16.   }
17.
18. }
```

Commentaires

- ligne 6 : la personne n'a qu'un attribut, son identifiant **[$id]**. On va supposer que celui-ci désigne une personne unique dans un entrepôt de données ;
- lignes 9-11 : le constructeur qui permet de construire un type **[Personne]** avec son identifiant ;
- lignes 14-16 : la méthode **[__toString]** qui affiche l'identifiant de la personne ;

1.10.3 Couche [dao]

L'interface **[IDao]** implémentée par la couche **[dao, 1]** est la suivante **[IDao.php]** :

```
1.  <?php
2.
3.  // couche [dao] ------------------------
4.  interface IDao {
5.
6.    // récupération d'une personne dans un entrepôt externe
7.    // on passe l'identifiant de la personne
8.    public function get(int $id): Personne;
9.
10.   // sauvegarde d'une personne dans un entrepôt externe
11.   public function save(Personne $p): void;
12. }
```

Cette interface est implémentée par la classe **[Dao1]** suivante **[Dao1.php]** :

```php
<?php

class Dao1 implements IDao {

  // sauvegarde d'une personne dans un entrepôt externe
  public function save(Personne $p): void {
    print "[Dao1] : Sauvegarde de la personne $p en base de données [locale]\n";
  }

  // récupération d'une personne dans un entrepôt externe
  public function get(int $id): Personne {
    print "[Dao1] : Récupération de la personne d'identité ($id) en base de données [locale]\n";
    return new Personne($id);
  }

}
```

Commentaires

- la classe n'interagit pas avec un entrepôt de données. On se contente d'afficher des messages pour suivre le déroulement du code (lignes 7 et 12) ;
- ligne 13 : on rend bien une personne ayant l'identifiant passé en paramètre à la méthode (ligne 11) ;

Nous implémentons également l'interface **[IDao]** avec la classe **[Dao2]** suivante **[Dao2.php]** :

```php
<?php

class Dao2 implements IDao {

  // sauvegarde d'une personne dans un entrepôt externe
  public function save(Personne $p): void {
    print "[Dao2] : Sauvegarde de la personne $p en base de données [distante]\n";
  }

  // récupération d'une personne dans un entrepôt externe
  public function get(int $id): Personne {
    print "[Dao2] : Récupération de la personne d'identité ($id) en base de données [distante]\n";
    return new Personne($id);
  }

}
```

La classe **[Dao2]** est similaire à la classe **[Dao1]** si ce n'est que nous avons modifié les messages affichés.

1.10.4 Couche [métier]

La couche **[métier, 2]** offre l'interface **[IMétier]** suivante **[IMetier.php]** :

```php
<?php

// couche [métier] ------------------------
interface IMétier extends IDao {

  // on exploite une personne identifiée par son id
  public function doSomething(Personne $p): void;
}
```

Commentaires

- l'interface **[IMétier]** étend l'interface **[IDao]**. Ce n'est pas du tout obligatoire. On le fait ici parce que l'exemple est simple ;
- ligne 7 : la méthode **[doSomething]** est propre à la couche **[Métier]** ;

L'interface **[IMétier]** sera implémentée par la classe **[Métier]** suivante **[Métier.php]** :

```php
<?php

class Métier implements IMétier {
  // couche [dao]
  private $dao;

  // getter / setter
```

```
8.    public function setDao(IDao $dao): void {
9.      $this->dao = $dao;
10.   }
11.
12.   public function getDao(): IDao {
13.     return $this->dao;
14.   }
15.
16.   // on exploite une personne
17.   public function doSomething(Personne $p): void {
18.     // traitement
19.     print "[Métier] : Traitement métier de la personne $p\n";
20.   }
21.
22.   // sauvegarde de la personne
23.   public function save(Personne $p): void {
24.     print "[Métier] : Sauvegarde de la personne $p\n";
25.     // on demande à la couche [dao] de faire la sauvegarde
26.     $this->dao->save($p);
27.   }
28.
29.   public function get(int $id): Personne {
30.     print "Métier : récupération de la personne d'identifiant $id\n";
31.     // on demande la personne à la couche [dao]
32.     $personne = $this->dao->get($id);
33.     return $personne;
34.   }
35.
36. }
```

Commentaires

- ligne 5 : la couche **[métier]** doit avoir une référence sur la couche **[dao]** pour pouvoir utiliser les méthodes de celle-ci ;
- lignes 8-10 : la méthode **[setDao]** permet de donner à la couche **[métier]** la référence de la couche **[dao]**. On notera que le type du paramètre est **[IDao]**. Ainsi la couche **[métier]** va être capable de travailler avec toute classe implémentant l'interface **[IDao]**. Si on passe d'une couche **[Dao1]** à une couche **[Dao2]** et que toutes deux implémentent l'interface **[IDao]**, la couche **[métier]** n'a pas à être réécrite ;
- lignes 17-20 : implémentation de **[IMetier::doSomething]** ;
- lignes 23-27 : implémentation de **[IMetier::save]**. Cette méthode doit sauvegarder une personne dans un entrepôt de données. La couche **[métier]** ne sait pas faire ça. Elle s'adresse à la couche **[dao]** pour faire cette sauvegarde ;
- lignes 29-34 : implémentation de **[IMetier::get]**. Cette méthode doit récupérer dans un entrepôt de données la personne dont l'identifiant lui est passé en paramètre. La couche **[métier]** ne sait pas faire ça. Elle s'adresse à la couche **[dao]** pour faire le travail (ligne 32) ;

Conclusion

Dès que la couche **[métier]** a besoin d'accéder aux données stockées dans l'entrepôt de données, elle doit passer par la couche **[dao]** qui a été créée pour accéder à ces données.

1.10.5 Script principal

Nous allons écrire deux scripts, chefs d'orchestre pour cette application. Le 1er **[main1.php]** utilisera la couche **[Dao1]** alors que le second **[main2.php]** utilisera la couche **[Dao2]**. Nous voulons montrer que cela n'a aucune incidence sur le code de la couche **[Métier]**.

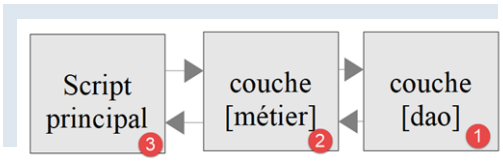
Le script **[main1.php]** est le suivant :

```
1.  <?php
2.
3.  // respect strict des paramètres des fonctions
4.  declare (strict_types=1);
5.
6.  // inclusion classes et interfaces
7.  require_once __DIR__."/Personne.php";
8.  require_once __DIR__."/IDao.php";
9.  require_once __DIR__."/IMetier.php";
10. require_once __DIR__."/Dao1.php";
11. require_once __DIR__."/Métier.php";
12.
13. // test ----------------
14. // création des couches
15. $dao1 = new Dao1();
16. $métier = new Métier();
```

```
17. $métier->setDao($dao1);
18. // utilisation de la couche [métier]
19. $personne = $métier->get(4);
20. $métier->doSomething($personne);
21. $métier->save($personne);
```

Commentaires

- rappelons quelques points : le script **[main1.php]** est le chef d'orchestre de l'application. Il crée la structure en couches de l'application (lignes 15-17) puis ensuite commence à dialoguer avec la couche **[métier]** (lignes 19-21). La structure en couches est en effet la suivante :



  Selon ce schéma, le script **[main1.php]** ne doit s'adresser qu'à la couche **[métier]**. Elle ne doit pas s'adresser à la couche **[dao]** même si c'est théoriquement possible.

Les résultats de l'exécution sont les suivants :

```
1   [Métier] : récupération de la personne d'identifiant 4
2   [Dao1] : Récupération de la personne d'identité (4) en base de données [locale]
3   [Métier] : Traitement métier de la personne [Personne(4)]
4   [Métier] : Sauvegarde de la personne [Personne(4)]
5   [Dao1] : Sauvegarde de la personne [Personne(4)] en base de données [locale]
```

Commentaires

- la ligne 10 du code a provoqué l'écriture des lignes 1 et 2 des résultats ;
- la ligne 20 du code a provoqué l'écriture de la ligne 3 des résultats ;
- la ligne 21 du code a provoqué l'écriture des lignes 4 et 5 des résultats ;

Le script **[main2.php]** est le suivant :

```
1.  <?php
2.
3.  // respect strict des paramètres des fonctions
4.  declare (strict_types=1);
5.
6.  // inclusion classes et interfaces
7.  require_once __DIR__."/Personne.php";
8.  require_once __DIR__."/IDao.php";
9.  require_once __DIR__."/IMetier.php";
10. require_once __DIR__."/Dao2.php";
11. require_once __DIR__."/Métier.php";
12.
13. // test ----------------
14. // création des couches
15. $dao2 = new Dao2();
16. $métier = new Métier();
17. $métier->setDao($dao2);
18. // utilisation de la couche [métier]
19. $personne = $métier->get(4);
20. $métier->doSomething($personne);
21. $métier->save($personne);
```

Commentaires

- lignes 15-17 : la structure en couches utilise désormais la couche **[Dao2]** ;

Les résultats de l'exécution sont les suivants :

```
1   [Métier] : récupération de la personne d'identifiant 4
2   [Dao2] : Récupération de la personne d'identité (4) en base de données [distante]
3   [Métier] : Traitement métier de la personne [Personne(4)]
4   [Métier] : Sauvegarde de la personne [Personne(4)]
```

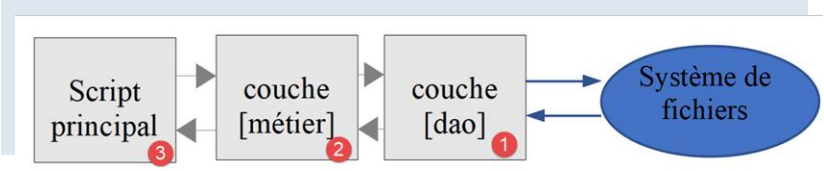
```
5   [Dao2] : Sauvegarde de la personne [Personne(4)] en base de données [distante]
```

La couche **[Dao1]** simule un accès à une base de données locale alors que la couche **[Dao2]** simule un accès à une base de données distante. Tant que ces deux couches respectent l'interface **[IDao]**, on voit que le code de la couche **[Métier]** n'a pas eu à être changé.

Nous allons appliquer ce que nous venons d'apprendre à l'exercice du calcul d'impôts qui nous sert de fil rouge.

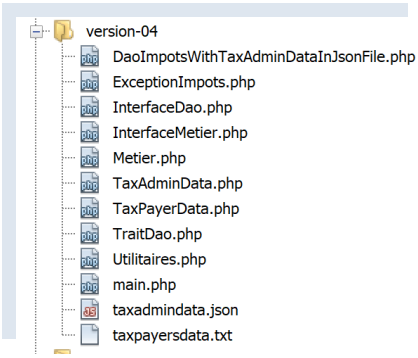
1.11 Exercice d'application – version 4

L'application de calcul d'impôts va implémenter la structure en couches suivante :



Nous allons reprendre les éléments de la version 3 du paragraphe lien en les modifiant pour les adapter à la nouvelle architecture de l'application. On appelle parfois cela du 'refactoring'. Nous supposons ici que les données nécessaires à l'application sont dans des fichiers texte. C'est la couche **[Dao]** qui va s'occuper des échanges avec ces fichiers.

1.11.1 Arborescence des scripts



1.11.2 Objets échangés entre couches

Nous allons garder certains objets de la version 3. Nous les redonnons ici pour rappel.

L'exception **[ExceptionImpots]** est l'exception que lancera la couche **[Dao]** lorsqu'elle rencontrera un problème soit avec l'accès aux données soit avec la nature des données (données incorrectes).

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  class ExceptionImpots extends \RuntimeException {
7.
8.    public function __construct(string $message, int $code=0) {
9.      parent::__construct($message, $code);
10.   }
11. }
```

La classe **[Utilitaires]** rassemble des méthodes utiles à la gestion des fichiers texte (ici une seule méthode) :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  // une classe de fonctions utilitaires
7.  abstract class Utilitaires {
8.
9.    public static function cutNewLinechar(string $ligne): string {
```

```
    // on supprime la marque de fin de ligne de $ligne si elle existe
    $longueur = strlen($ligne);  // longueur ligne
    while (substr($ligne, $longueur - 1, 1) == "\n" or substr($ligne, $longueur - 1, 1) == "\r") {
      $ligne = substr($ligne, 0, $longueur - 1);
      $longueur--;
    }
    // fin - on rend la ligne
    return($ligne);
  }
}
```

La classe **[TaxAdminData]** est la classe qui encapsule les données de l'administration fiscale :

```
<?php

namespace Application;

class TaxAdminData {
  // tranches d'impôt
  private $limites;
  private $coeffR;
  private $coeffN;
  // constantes de calcul de l'impôt
  private $plafondQfDemiPart;
  private $plafondRevenusCelibatairePourReduction;
  private $plafondRevenusCouplePourReduction;
  private $valeurReducDemiPart;
  private $plafondDecoteCelibataire;
  private $plafondDecoteCouple;
  private $plafondImpotCouplePourDecote;
  private $plafondImpotCelibatairePourDecote;
  private $abattementDixPourcentMax;
  private $abattementDixPourcentMin;

  // initialisation
  public function setFromJsonFile(string $taxAdminDataFilename): TaxAdminData {
    // on récupère le contenu du fichier des données fiscales
    $fileContents = \file_get_contents($taxAdminDataFilename);
    …
    // on rend l'objet
    return $this;
  }

  private function check($value): \stdClass {
    …
    return $result;
  }

    // toString
  public function __toString() {
    // chaîne Json de l'objet
    return \json_encode(\get_object_vars($this), JSON_UNESCAPED_UNICODE);
  }

  // getters et setters
  public function getLimites() {
    return $this->limites;
  }

  …

  public function setLimites($limites) {
    $this->limites = $limites;
    return $this;
  }
…

}
```

Nous ajoutons une nouvelle classe **[TaxPayerData]** qui encapsule les données écrites dans le fichier des résultats :

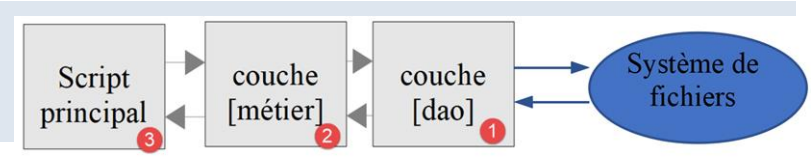
```
<?php

// espace de noms
namespace Application;

```

```php
// la classe des données
class TaxPayerData {
  // données nécessaires au calcul de l'impôt du contribuable
  private $marié;
  private $enfants;
  private $salaire;
  // résultats du calcul de l'impôt
  private $montant;
  private $surcôte;
  private $décôte;
  private $réduction;
  private $taux;

  // setter
  public function setFromParameters(string $marié, int $nbEnfants, int $salaireAnnuel) : TaxPayerData{
    // données du contribuable nécessaires au calcul de l'impôt
    $this->marié = $marié;
    $this->enfants = $nbEnfants;
    $this->salaire = $salaireAnnuel;
    // on rend l'objet initialisé
    return $this;
  }

  // getters et setters
  public function getMarié() {
    return $this->marié;
  }

  …

  public function setMarié($marié) {
    $this->marié = $marié;
    return $this;
  }

  …

    // toString
  public function __toString() {
    // chaîne Json de l'objet
    return \json_encode(\get_object_vars($this), JSON_UNESCAPED_UNICODE);
  }

}
```

Note : utilisez la génération automatique de code pour générer le constructeur, les getters et setters (cf paragraphe lien). Remarquez que les setters sont 'fluents'.

1.11.3 La couche `[dao]`

Nous nous intéressons ici à la couche **`[1]`** de notre application :



1.11.3.1 L'interface `[InterfaceDao]`

L'interface de la couche **`[dao]`** sera la suivante **`[InterfaceDao.php]`** :

```php
<?php

// espace de noms
namespace Application;

interface InterfaceDao {

  // lecture des données contribuables
  public function getTaxPayersData(string $taxPayersFilename, string $errorsFilename): array;

```

```
11.   // lecture des données de l'administration fiscale (tranches d'impôts)
12.   public function getTaxAdminData(): TaxAdminData;
13.
14.   // enregistrement des résultats
15.   public function saveResults(string $resultsFilename, array $taxPayersData): void;
16. }
```

Commentaires

- le cahier des charges est ici le suivant :
  - les données des contribuables sont dans un fichier texte ;
  - on enregistre les résultats du calcul d'impôts dans un fichier texte ;
  - on enregistre les éventuelles erreurs dans un fichier texte ;
  - on ne sait pas sous quelle forme sont disponibles les données de l'administration fiscale. Pour chaque nouvelle forme, l'interface **[InterfaceDao]** devra être implémentée par une nouvelle classe ;
  - les méthodes de l'interface qui rencontrent une erreur irrécupérable lors de l'accès aux données doivent lancer une exception de type **[ExceptionImpots]** ;
- ligne 9 : la méthode qui permet d'obtenir les données du contribuable **[statut marital, nombre d'enfants, salaire annuel]** ;
  - le 1er paramètre est le nom du fichier texte dans lequel se trouvent ces données ;
  - le second paramètre est le nom du fichier texte dans lequel enregistrer les éventuelles erreurs rencontrées ;
- ligne 12 : la méthode qui permet d'obtenir les données de l'administration fiscale. On ne lui passe ici aucun paramètre car on ne sait pas comment elles sont stockées ;
- ligne 15 : la méthode qui permet d'enregistrer les résultats du calcul de l'impôt dans un fichier texte dont on passe le nom en paramètre ;

Lorsqu'on écrit l'interface **[InterfaceDao]**, on sait qu'il y aura différentes façons d'écrire la méthode **[getTaxAdminData]** selon la façon dont seront stockées les données de l'administration fiscale. L'interface **[InterfaceDao]** sera donc implémentée par différentes classes, chacune s'occupant d'un stockage particulier de ces données (tableaux, fichiers texte, base de données, service web). Ces classes dérivées auront néanmoins un code commun, celui de l'implémentation des méthodes **[getTaxPayersData, saveResults]**. On sait que ce cas d'utilisation peut être implémenté de deux façons (cf paragraphe lien):

1. on crée une classe **abstraite C** qui regroupe le code commun aux classes dérivées. La classe C implémente l'interface I mais certaines méthodes qui doivent être déclarées dans les classes dérivées sont dans la classe C déclarées abstraites et donc la classe C est elle-même abstraite. On crée ensuite des classes C1 et C2 dérivées de C qui implémentent chacune à leur manière les méthodes non définies (abstraites) de leur classe parent C ;
2. on crée un **trait T** quasi identique à la classe abstraite C de la solution précédente. Ce trait n'implémente pas l'interface I car syntaxiquement elle ne le peut pas. On crée ensuite des classes C1 et C2 implémentant l'interface I et utilisant le trait T. Il ne reste à ces classes qu'à implémenter les méthodes de l'interface I non implémentées par le trait T qu'elles importent ;

Pour l'exemple, nous allons utiliser ici un trait **[TraitDao]**.

1.11.3.2 Le trait [TraitDao]

Le code du trait **[TraitDao]** est le suivant **[TraitDao.php]** :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  trait TraitDao {
7.
8.    // lecture des données contribuables
9.    public function getTaxPayersData(string $taxPayersFilename, string $errorsFilename): array {
10.     // tableau des données contribuables
11.     $taxPayersData = [];
12.     // tableau des erreurs
13.     $errors = [];
14.     // pas mal d'erreurs peuvent se produire dès qu'on gère des fichiers
15.     try {
16.       // lecture des données utilisateur
17.       // chaque ligne a la forme statut marital, nombre d'enfants, salaire annuel
18.       $taxPayersFile = fopen($taxPayersFilename, "r");
19.       if (!$taxPayersFile) {
20.         throw new ExceptionImpots("Impossible d'ouvrir en lecture les déclarations des contribuables [$taxPayersFilename
    ]", 12);
21.       }
22.       // on exploite la ligne courante du fichier des données utilisateur
23.       // qui a la forme statut marital, nombre d'enfants, salaire annuel
24.       $num = 1;         // n° ligne courante
25.       $nbErreurs = 0;   // nbre d'erreurs rencontrées
26.       while ($ligne = fgets($taxPayersFile, 100)) {
27.         // on néglige les lignes vides
```

```
28.         $ligne = trim($ligne);
29.         if (strlen($ligne) == 0) {
30.           // ligne suivante
31.           $num++;
32.           // on reboucle
33.           continue;
34.         }
35.         // on enlève l'éventuelle marque de fin de ligne
36.         $ligne = Utilitaires::cutNewLineChar($ligne);
37.         // on récupère les 3 champs marié:enfants:salaire qui forment $ligne
38.         list($marié, $enfants, $salaire) = explode(",", $ligne);
39.         // on les vérifie
40.         // le statut marital doit être oui ou non
41.         $marié = trim(strtolower($marié));
42.         $erreur = ($marié !== "oui" and $marié !== "non");
43.         if (!$erreur) {
44.           // le nombre d'enfants doit être un entier
45.           $enfants = trim($enfants);
46.           if (!preg_match("/^\d+$/", $enfants)) {
47.             $erreur = TRUE;
48.           } else {
49.             $enfants = (int) $enfants;
50.           }
51.         }
52.         if (!$erreur) {
53.           // le salaire est un entier sans les centimes d'euros
54.           $salaire = trim($salaire);
55.           if (!preg_match("/^\d+$/", $salaire)) {
56.             $erreur = TRUE;
57.           } else {
58.             $salaire = (int) $salaire;
59.           }
60.         }
61.         // erreur ?
62.         if ($erreur) {
63.           $errors[] = "la ligne [$num] du fichier [$taxPayersFilename] est erronée";
64.           $nbErreurs++;
65.         } else {
66.           // on mémorise les informations
67.           $taxPayersData[] = (new TaxPayerData())->setFromParameters($marié, $enfants, $salaire);
68.         }
69.         // ligne suivante
70.         $num++;
71.       }
72.       // est-on à la fin du fichier ?
73.       if (!feof($taxPayersFile)) {
74.         // on est sorti de la boucle sur une erreur de lecture
75.         throw new ExceptionImpots("Erreur lors de la lecture de la ligne n° [$num] du fichier [$taxPayersFilename]");
76.       } else {
77.         // on est sorti de la boucle sur la marque de fin de fichier
78.         // on sauve les erreurs dans un fichier texte
79.         $this->saveString($errorsFilename, implode("\n", $errors));
80.         // résultat de la fonction
81.         return $taxPayersData;
82.       }
83.     } finally {
84.       // on ferme le fichier s'il est ouvert
85.       if ($taxPayersFile) {
86.         fclose($taxPayersFile);
87.       }
88.     }
89.   }
90.
91.   // enregistrement des résultats
92.   public function saveResults(string $resultsFilename, array $taxPayersData): void {
93.     // enregistrement du tableau [$taxPayersData] dans le fichier texte [$resultsFileName]
94.     // si le fichier texte [$resultsFileName] n'existe pas, il est créé
95.     $this->saveString($resultsFilename, implode("\n", $taxPayersData));
96.   }
97.
98.   // enregistrement d'es résultats d'un tableau dans un fichier texte
99.   private function saveString(string $fileName, string $data): void {
100.    // enregistrement du tableau [$data] dans le fichier texte [$fileName]
101.    // si le fichier texte [$fileName] n'existe pas, il est créé
102.    if (file_put_contents($fileName, $data) === FALSE) {
103.      throw new ExceptionImpots("Erreur lors de l'enregistrement de données dans le fichier texte [$fileName]");
104.    }
105.  }
106.
107. }
```

Commentaires

- ligne 6 : nous définissons ici un **trait** et non une **classe** ;

- lignes 9-89 : la méthode **[getTaxPayersData]** implémente la méthode de même nom de l'interface **[InterfaceDao]**. Elle récupère dans un fichier texte nommé **[$taxPayersFilename]** les données des contribuables **[statut marital, nombre d'enfants, salaire annuel]**. Elle rend celles-ci sous la forme d'un tableau **[$taxPayersData]** d'éléments de type **[TaxPayerData]** (lignes 67, 81) ;
- la méthode **[getTaxPayersData]** est très semblable à la méthode **[AbstractBaseImpots::executeBatchImpots]** décrite au paragraphe lien avec les différences suivantes :
  - la méthode **[getTaxPayersData]** ne fait que récupérer les données des contribuables. **Elle ne fait pas de calcul d'impôt**. Ici c'est le rôle de la couche **[métier]** ;
  - comme le faisait la méthode **[executeBatchImpots]** elle signale les erreurs. Ici les erreurs sont d'abord mémorisées dans un tableau **[$errors]** (ligne 13), tableau qui est mémorisé dans un fichier texte à la fin du traitement (ligne 79). Selon les cas, il est vide ou non ;
  - dans le cas d'erreur irrécupérable, une exception de type **[ExceptionImpots]** est lancée (lignes 20, 75) ;
- ligne 73 : on notera le traitement fait à la sortie de la boucle des lignes 26-71. En effet la fonction **[fgets]** a l'inconvénient de rendre le booléen FALSE aussi bien lorsque la lecture des lignes a rencontré la marque de fin de fichier que si cette lecture n'a pu aboutir à cause d'une erreur. Pour différentier les deux cas, on teste si on est rendu à la fin du fichier avec la fonction **[feof]**. Si on n'est pas rendu à la fin du fichier, c'est qu'une erreur s'est produite et on lance alors une exception ;
- lignes 83-88 : le **[finally]** est exécuté qu'il y ait eu exception ou pas lors de l'exploitation du fichier ;
- ligne 85 : si le fichier a été ouvert, alors le 'handle' **[$taxPayersFile]** du fichier a la valeur booléenne TRUE, FALSE sinon ;
- lignes 99-105 : la méthode privée **[saveString]** utilisée ligne 79 pour enregistrer le tableau des erreurs dans un fichier texte ;
- ligne 99 : la méthode **[saveString]** reçoit deux paramètres :
  - **[string $filename]** qui est le nom du fichier texte utilisé pour enregistrer les données ;
  - **[string $data]** qui est la chaîne de caractères à enregistrer dans le fichier texte. Cette chaîne sera un ensemble de lignes terminée par le caractère de fin de ligne \n ;
- ligne 102 : la fonction PHP **[file_puts_contents]** enregistre une chaîne de caractères dans un fichier texte. Elle s'occupe d'ouvrir le fichier, d'écrire la chaîne dedans et de fermer le fichier. Elle rend le booléen FALSE si une erreur s'est produite ;
- ligne 103 : si une erreur se produit, on lance une exception ;
- lignes 92-96 : implémentation de la méthode **[saveResults]** de l'interface **[InterfaceDao]**. On utilise de nouveau la méthode privée **[saveString]**. Ici le second paramètre de **[saveString]** est une chaîne construite à partir du tableau **[$taxPayersData]** dont les éléments sont de type **[TaxPayerData]**. On peut se demander quel va être le résultat de l'opération :

```
implode("\n", $taxPayersData)
```

Nous avons défini dans la classe **[TaxPayerData]** (paragraphe lien) la méthode **[__toString]** suivante :

```
1   public function __toString() {
2       // chaîne Json de l'objet
3       return \json_encode(\get_object_vars($this), JSON_UNESCAPED_UNICODE);
4    }
```

L'opération

```
implode("\n", $taxPayersData)
```

va concaténer chaque élément du tableau **[$taxPayersData]** transformé en chaîne de caractères par sa méthode **[__toString]** avec la marque de fin de ligne \n. Cela va donner une chaîne de caractères de la forme :

json1\njson2\n…

Conclusion

Le trait **[TraitDao]** a implémenté deux des méthodes de l'interface **[InterfaceDao]**, **[getTaxPayersData]** et **[saveResults]** :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  interface InterfaceDao {
7.
8.    // lecture des données contribuables
9.    public function getTaxPayersData(string $taxPayersFilename, string $errorsFilename): array;
10.
11.   // lecture des données de l'administration fiscale (tranches d'impôts)
12.   public function getTaxAdminData(): TaxAdminData;
13.
14.   // enregistrement des résultats
15.   public function saveResults(string $resultsFilename, array $taxPayersData): void;
16. }
```

Il nous reste à implémenter la méthode **[getTaxAdminData]** qui récupère les données de l'administration fiscale.

1.11.3.3 La classe [DaoImpotsWithTaxAdminDataInJsonFile]

La classe **[DaoImpotsWithTaxAdminDataInJsonFile]** implémente l'interface **[InterfaceDao]** de la façon suivante :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  // définition d'une classe ImpotsWithDataInFile
7.  class DaoImpotsWithTaxAdminDataInJsonFile implements InterfaceDao {
8.    // usage d'un trait
9.    use TraitDao;
10.   // l'objet de type TaxAdminData qui contient les données des tranches d'impôts
11.   private $taxAdminData;
12.
13.   // le constructeur
14.   public function __construct(string $taxAdminDataFilename) {
15.     // on veut initialiser l'attribut [$this->taxAdminData]
16.     $this->taxAdminData = (new TaxAdminData())->setFromJsonFile($taxAdminDataFilename);
17.   }
18.
19.   // retourne les données permettant le calcul de l'impôt
20.   public function getTaxAdminData(): TaxAdminData {
21.     return $this->taxAdminData;
22.   }
23. }
```

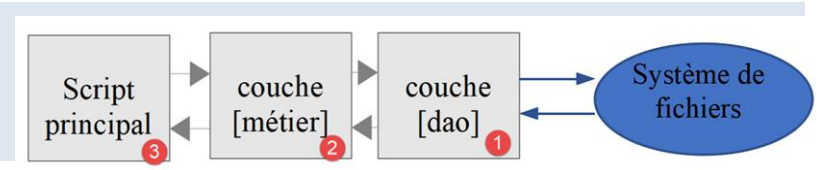
Commentaires

- ligne 7 : la classe **[ImpotsWithTaxAdminDataInJsonFile]** implémente l'interface **[InterfaceDao]** ;
- ligne 9 : la classe **[ImpotsWithTaxAdminDataInJsonFile]** utilise le trait **[traitDao]** qui on le sait implémente les méthodes **[getTaxPayersData]** et **[saveResults]** de l'interface **[InterfaceDao]**. Il ne reste donc plus, à la classe **[ImpotsWithTaxAdminDataInJsonFile]** qu'à implémenter la méthode **[getTaxAdminData]** qui récupère les données de l'administration fiscale ;
- ligne 11 : l'attribut de type **[TaxAdminData]** que rend la méthode **[getTaxAdminData]** des lignes 20-22. Cet attribut est initialisé par le constructeur des lignes 14-17 ;

Nous en avons terminé avec la couche **[dao]** de notre application : nous avons une classe qui implémente totalement l'interface **[InterfaceDao]** que nous nous sommes imposés. Nous pouvons désormais passer à la couche**[métier]**.

1.11.4 La couche [métier]

Nous allons maintenant implémenter la couche **[2]** de notre architecture :



1.11.4.1 L'interface [InterfaceMétier]

L'interface de la couche **[métier]** sera la suivante :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  interface InterfaceMetier {
7.
8.    // calcul des impôts d'un contribuable
9.    public function calculerImpot(string $marié, int $enfants, int $salaire): array;
10.
```

```
11.    // calcul des impôts en mode batch
12.    public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string $errorsFile
    Name): void;
13. }
```

Commentaires

- ligne 9 : l'interface **[InterfaceMétier]** sait calculer le montant de l'impôt d'un contribuable particulier pourvu qu'on lui donne les informations suivantes : statut marital, nombre d'enfants, salaire annuel. La méthode **[calculerImpot]** n'utilise pas la couche **[dao]** aussi ne lance-t-elle pas d'exceptions ;
- ligne 9 : l'interface **[InterfaceMétier]** peut aussi calculer le montant de l'impôt d'un ensemble de contribuables dont les données sont rassemblées dans le fichier texte nommé **[$taxPayersFileName]**. Elle met les résultats dans un fichier texte nommé **[$resultsFileName]**. La méthode **[executeBatchImpots]** doit s'adresser à la couche **[dao]** qui s'occupe des accès au système de fichiers. Des exceptions peuvent alors remonter de la couche **[dao]** que la méthode **[executeBatchImpots]** n'interceptera pas : elle les laissera remonter au script principal. Les erreurs non fatales sont enregistrées dans le fichier texte nommé **[$errorsFileName]** ;
- ligne 9 : la méthode **[calculerImpot]** est une méthode purement **[métier]**. Elle ne se préoccupe pas d'où viennent les données qu'elle utilise ;
- ligne 12 : la méthode **[executeBatchImpots]** va s'adresser à la couche **[dao]** pour lire et écrire des données dans des fichiers texte. Elle va appeler de façon répétée la méthode métier **[calculerImpot]** ;

1.11.4.2 La classe [Metier]

La classe **[Metier]** implémente l'interface **[InterfaceMetier]** de la façon suivante :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  class Metier implements InterfaceMetier {
7.    // couche Dao
8.    private $dao;
9.    // données administration fiscale
10.   private $taxAdminData;
11.
12.   //---------------------------------------------
13.   // setter couche [dao]
14.   public function setDao(InterfaceDao $dao) {
15.     $this->dao = $dao;
16.     return $this;
17.   }
18.
19.   public function __construct(InterfaceDao $dao) {
20.     // on mémorise une référence sur la couche [dao]
21.     $this->dao = $dao;
22.     // on récupère les données permettant le calcul de l'impôt
23.     // la méthode [getTaxAdminData] peut lancer une exception ExceptionImpots
24.     // on la laisse alors remonter au code appelant
25.     $this->taxAdminData = $this->dao->getTaxAdminData();
26.   }
27.
28. // calcul de l'impôt
29. // ------------------------------------------------------------------------
30.   public function calculerImpot(string $marié, int $enfants, int $salaire): array {
31.     …
32.     // résultat
33.     return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" => $réduction,
    "taux" => $taux];
34.   }
35.
36. // ------------------------------------------------------------------------
37.   private function calculerImpot2(string $marié, int $enfants, float $salaire): array {
38.     …
39.     // résultat
40.     return ["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]];
41.   }
42.
43.   // revenuImposable=salaireAnnuel-abattement
44.   // l'abattement a un min et un max
45.   private function getRevenuImposable(float $salaire): float {
46.     …
47.     // résultat
48.     return floor($revenuImposable);
49.   }
50.
51. // calcule une décôte éventuelle
52.   private function getDecôte(string $marié, float $salaire, float $impots): float {
53.     …
54.     // résultat
```

```
55.       return ceil($décôte);
56.     }
57.
58.   // calcule une réduction éventuelle
59.     private function getRéduction(string $marié, float $salaire, int $enfants, float $impots): float {
60.       …
61.       // résultat
62.       return ceil($réduction);
63.     }
64.
65.     // calcul des impôts en mode batch
66.     public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string $errorsFileName): void {
67.       …
68.       // enregistrement des résultats
69.       $this->dao->saveResults($resultsFileName, $results);
70.     }
71.
72.   }
```

Commentaires

- ligne 6 : la classe **[Metier]** implémente l'interface **[InterfaceMetier]**, ç-à-d les méthodes **[calculerImpot]** (lignes 30-34) et **[executeBatchImpots]** (lignes 66-70) ;
- ligne 8 : une référence sur la couche **[dao]**. Il en faut obligatoirement une pour que la couche **[métier]** sache à qui s'adresser lorsqu'elle veut des données externes. Cet attribut sera initialisé via le setter des lignes 14-17 ou via le constructeur des lignes 19-26 ;
- ligne 10 : l'objet de type **[TaxAdminData]** qui encapsule les données de l'administration fiscale. Ces données sont nécessaires à la méthode métier **[calculerImpot]**. Cet attribut est initialisé via le constructeur des lignes 19-26 ;
- lignes 19-26 : le constructeur initialise les deux attributs de la classe :
  - l'attribut **[$dao]** est initialisé avec la référence passée en paramètre au constructeur. On notera que le type de ce paramètre est celui de l'interface **[InterfaceDao]** permettant ainsi à la classe **[Metier]** d'être initialisée par n'importe quelle classe implémentant cette interface ;
  - l'attribut **[$taxAdminData]** est initialisé en faisant appel à la méthode **[getTaxAdminData]** de la couche **[dao]** ;

On en conclut que lorsque les méthodes **[calculerImpots]** et **[executeBatchImpots]** s'exécutent, les deux attributs **[$dao]** et **[$taxAdminData]** sont initialisés.

La méthode **[calculerImpots]** est la suivante :

```
1.     public function calculerImpot(string $marié, int $enfants, int $salaire): array {
2.       // $marié : oui, non
3.       // $enfants : nombre d'enfants
4.       // $salaire : salaire annuel
5.       // $this->taxAdminData : données de l'administration fiscale
6.       //
7.       // on vérifie qu'on a bien les données de l'administration fiscale
8.       if ($this->taxAdminData === NULL) {
9.         $this->taxAdminData = $this->getTaxAdminData();
10.      }
11.      // calcul de l'impôt avec enfants
12.      $result1 = $this->calculerImpot2($marié, $enfants, $salaire);
13.      $impot1 = $result1["impôt"];
14.      // calcul de l'impôt sans les enfants
15.      if ($enfants != 0) {
16.        $result2 = $this->calculerImpot2($marié, 0, $salaire);
17.        $impot2 = $result2["impôt"];
18.        // application du plafonnement du quotient familial
19.        $plafonDemiPart = $this->taxAdminData->getPlafondQfDemiPart();
20.        if ($enfants < 3) {
21.          // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants
22.          $impot2 = $impot2 - $enfants * $plafonDemiPart;
23.        } else {
24.          // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants, le double pour les suivants
25.          $impot2 = $impot2 - 2 * $plafonDemiPart - ($enfants - 2) * 2 * $plafonDemiPart;
26.        }
27.      } else {
28.        $impot2 = $impot1;
29.        $result2 = $result1;
30.      }
31.      // on prend l'impôt le plus fort
32.      if ($impot1 > $impot2) {
33.        $impot = $impot1;
34.        $taux = $result1["taux"];
35.        $surcôte = $result1["surcôte"];
36.      } else {
37.        $surcôte = $impot2 - $impot1 + $result2["surcôte"];
38.        $impot = $impot2;
39.        $taux = $result2["taux"];
40.      }
41.      // calcul d'une éventuelle décôte
```

```
42.      $décôte = $this->getDecôte($marié, $salaire, $impot);
43.      $impot -= $décôte;
44.      // calcul d'une éventuelle réduction d'impôts
45.      $réduction = $this->getRéduction($marié, $salaire, $enfants, $impot);
46.      $impot -= $réduction;
47.      // résultat
48.      return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" => $réduction,
     "taux" => $taux];
49.    }
```

Commentaires

- ce code est celui de la méthode **[AbstractBaseImpots::calculerImpot]** de la version 3, expliquée au paragraphe lien. Il en est de même pour les méthodes privées **[calculerImpot2, getDecôte, getRéduction, getRevenuImposable]** ;

La méthode **[Metier::executeBatchImpots]** est la suivante :

```
1.     public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string $errorsFileName): void {
2.       // on laisse remonter les exceptions qui proviennent de la couche [dao]
3.       // on récupère les données contribuables
4.       $taxPayersData = $this->dao->getTaxPayersData($taxPayersFileName, $errorsFileName);
5.       // tableau des résultats
6.       $results = [];
7.       // on les exploite
8.       foreach ($taxPayersData as $taxPayerData) {
9.         // on calcule l'impôt
10.        $result = $this->calculerImpot(
11.          $taxPayerData->getMarié(),
12.          $taxPayerData->getEnfants(),
13.          $taxPayerData->getSalaire());
14.        // on complète [$taxPayerData]
15.        $taxPayerData->setMontant($result["impôt"]);
16.        $taxPayerData->setDécôte($result["décôte"]);
17.        $taxPayerData->setSurCôte($result["surcôte"]);
18.        $taxPayerData->setTaux($result["taux"]);
19.        $taxPayerData->setRéduction($result["réduction"]);
20.        // on met le résultat dans le tableau des résultats
21.        $results [] = $taxPayerData;
22.      }
23.      // enregistrement des résultats
24.      $this->dao->saveResults($resultsFileName, $results);
25.    }
```

Commentaires

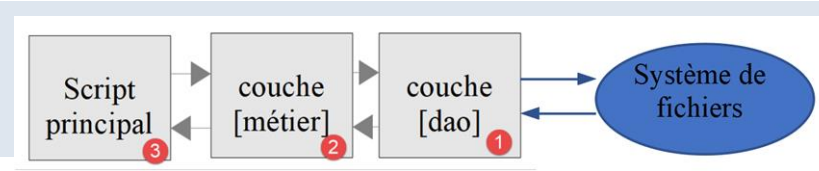
- ligne 1 : la méthode doit appeler de façon répétée la méthode **[calculerImpot]** pour chacun des contribuables trouvés dans le fichier texte nommé **[$taxPayersFileName]**. Elle doit mettre les résultats dans le fichier texte nommé **[$resultsFileName]**. Les erreurs non fatales rencontrées sont enregistrées dans dans le fichier texte nommé **[$errorsFileName]**. La méthode ne lance pas d'exceptions elle-même mais laisse remonter celles que la couche **[dao]** lance ;
- ligne 4 : les données des contribuables sont demandées à la couche **[dao]**. Celle-ci renvoie un tableau d'élements de type **[TaxPayerData]** qui est une classe d'attributs **[marié, nbEnfants, salaire, montant, décôte, réduction, surcôte, taux]** (cf paragraphe lien). S'il se produit une exception ici, comme elle n'est pas interceptée par un *catch*, elle remontera automatiquement au code appelant. Cela signifie qu'en cas d'exception, la ligne 6 n'est pas exécutée ;
- ligne 6 : le tableau des résultats de type **[TaxPayerData]** ;
- lignes 8-22 : on calcule l'impôt pour chacun des éléments du tableau des contribuables **[$taxPayersData]**. Pour cela, on fait appel à la méthode interne **[calculerImpot]** (ligne 10) ;
- lignes 15-19 : le résultat obtenu est utilisé pour initialiser les attributs de **[TaxPayerData]** qui ne l'étaient pas encore ;
- ligne 21 : le résultat obtenu est cumulé dans le tableau des résultats **[$results]** ;
- ligne 24 : une fois l'impôt calculé pour tous les contribuables, les résultats sont mémorisés dans un fichier texte. C'est la couche **[dao]** qui fait ce travail ;

Conclusion

En général la couche **[métier]** est assez simple à écrire car elle s'adresse à la couche **[dao]** qui, elle, gère l'accès aux données avec la gestion des erreurs qui va avec.

1.11.5 Le script principal

On écrit maintenant le script de la couche **[3]** de notre architecture :



Le script principal est le suivant **[main.php]** :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Application;
8.
9.  // gestion des erreurs par PHP
10. ini_set("display_errors", "0");
11.
12. // inclusion interface et classes
13. require_once __DIR__ . "/TaxAdminData.php";
14. require_once __DIR__ . "/TaxPayerData.php";
15. require_once __DIR__ . "/ExceptionImpots.php";
16. require_once __DIR__ . "/Utilitaires.php";
17. require_once __DIR__ . "/InterfaceDao.php";
18. require_once __DIR__ . "/TraitDao.php";
19. require_once __DIR__ . "/DaoImpotsWithTaxAdminDataInJsonFile.php";
20. require_once __DIR__ . "/InterfaceMetier.php";
21. require_once __DIR__ . "/Metier.php";
22. // test ----------------------------------------------------
23. // définition des constantes
24. const TAXPAYERSDATA_FILENAME = "taxpayersdata.txt";
25. const RESULTS_FILENAME = "resultats.txt";
26. const ERRORS_FILENAME = "errors.txt";
27. const TAXADMINDATA_FILENAME = "taxadmindata.json";
28.
29. try {
30.   // création de la couche [dao]
31.   $dao = new DaoImpotsWithTaxAdminDataInJsonFile(TAXADMINDATA_FILENAME);
32.   // création de la couche [métier]
33.   $métier = new Metier($dao);
34.   // calcul de l'impôts en mode batch
35.   $métier->executeBatchImpots(TAXPAYERSDATA_FILENAME, RESULTS_FILENAME, ERRORS_FILENAME);
36. } catch (ExceptionImpots $ex) {
37.   // on affiche l'erreur
38.   print $ex->getMessage() . "\n";
39. }
40. // fin
41. print "Terminé\n";
42. exit;
```

Commentaires

- ligne 24 : le nom du fichier des données contribuables ;
- ligne 25 : le nom du fichier des résultats ;
- ligne 26 : le nom du fichier des erreurs ;
- ligne 27 : le nom du fichier jSON contenant les données de l'administration fiscale;
- ligne 31 : création de la couche **[dao]** ;
- ligne 33 : création de la couche **[métier]** s'appuyant sur cette couche **[dao]** ;
- ligne 35 : exécution de la méthode **[executeBatchImpots]** de la couche **[métier]** ;
- lignes 36-39 : on a vu que la couche **[métier]** pouvait remonter des exceptions. Elles sont interceptées ici ;

1.11.6 Tests visuels

1.11.6.1 Test n° 1

Avec le fichier des contribuables **[taxpayersdata.txt]** suivants :

```
1   oui,2,55555
2   oui,2,50000
3   oui,3,50000
```

```
4   non,2,100000
5   non,3X,100000
6   oui,3,100000
7   oui,5,100000X
8   non,0,100000
9   oui,2,30000
10  non,0,200000
11  oui,3,200000
```

on obtient le fichier des erreurs **[errors.txt]** suivant :

```
1   la ligne [5] du fichier [taxpayersdata.txt] est erronée
2   la ligne [7] du fichier [taxpayersdata.txt] est erronée
```

et le fichier des résultats **[resultats.txt]** suivant :

```
1   {"marié":"oui","enfants":2,"salaire":55555,"impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.14}
2   {"marié":"oui","enfants":2,"salaire":50000,"impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.14}
3   {"marié":"oui","enfants":3,"salaire":50000,"impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14}
4   {"marié":"non","enfants":2,"salaire":100000,"impôt":19884,"surcôte":4480,"décôte":0,"réduction":0,"taux":0.41}
5   {"marié":"oui","enfants":3,"salaire":100000,"impôt":9200,"surcôte":2180,"décôte":0,"réduction":0,"taux":0.3}
6   {"marié":"non","enfants":0,"salaire":100000,"impôt":22986,"surcôte":0,"décôte":0,"réduction":0,"taux":0.41}
7   {"marié":"oui","enfants":2,"salaire":30000,"impôt":0,"surcôte":0,"décôte":0,"réduction":0,"taux":0}
8   {"marié":"non","enfants":0,"salaire":200000,"impôt":64210,"surcôte":7498,"décôte":0,"réduction":0,"taux":0.45}
9   {"marié":"oui","enfants":3,"salaire":200000,"impôt":42842,"surcôte":17283,"décôte":0,"réduction":0,"taux":0.41}
```

1.11.6.2 Test n° 2

Dans le script principal, on met pour le fichier des contribuables un nom de fichier qui n'existe pas :

```
const TAXPAYERS_DATA_FILENAME = "taxpayersdata2.txt";
```

Les résultats obtenus à la console alors sont les suivants :

```
1   Warning: fopen(taxpayersdata2.txt): failed to open stream: No such file or directory in C:\Data\st-
    2019\dev\php7\poly\scripts-console\impots\version-04\TraitDao.php on line 18
2   Impossible d'ouvrir en lecture les déclarations des contribuables [taxpayersdata2.txt]
3   Terminé
4   Done.
```

- ligne 1 : avertissements (warning) de l'interpréteur PHP ;
- ligne 2 : le message d'erreur de l'exception lancée par la couche **[dao]** ;

Il est possible de mettre en sourdine les messages d'erreur de l'interpréteur PHP :

```
16  require_once __DIR__ . "/ImpotsWithTaxAdminDataInArraysDao.php";
17  require_once __DIR__ . "/InterfaceMetier.php";
18  require_once __DIR__ . "/Metier.php";
19
20  // gestion des erreurs par PHP
21  ini_set("display_errors", "0");
22
23  // test ----------
24  // définition des constantes
25  const TAXPAYERS_DATA_FILENAME = "taxpayersdata2.txt";
```

La ligne 21 du code ci-dessus demande à ce que les erreurs PHP ne soient pas affichées. Pendant la phase de développement il est nécessaire qu'elles soient affichées. En mode production, il faut les cacher.

Les résultats de l'exécution sont alors les suivants :

```
1   Impossible d'ouvrir en lecture les déclarations des contribuables [taxpayersdata2.txt]
2   Terminé
```

1.11.7 Tests [Codeception]

Les tests visuels sont très insuffisants :

- on se limite en général à quelques tests ;
- on est plus ou moins attentif lors de cette vérification visuelle et des détails peuvent nous échapper ;

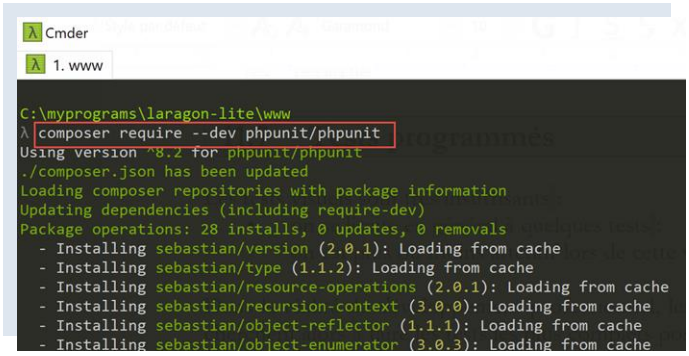
Dans la réalité du développement professionnel, les tests sont rédigés par des personnes dédiées dont c'est le rôle principal. Elles cherchent alors à faire les tests les plus complets possible(s). Pour cela elles utilisent des frameworks de test.

Nous allons ici utiliser le framework **Codeception [https://codeception.com/]** car il peut être intégré à Netbeans. C'est un framework avec un large éventail de possibilités. Nous n'allons en utiliser que quelques-unes. L'idée est d'avoir un moyen rapide, après chaque nouvelle version de l'exercice d'application, de vérifier que celle-ci fonctionne. L'existence de tests réussis donne au développeur confiance dans le code qu'il a écrit. C'est un facteur important.

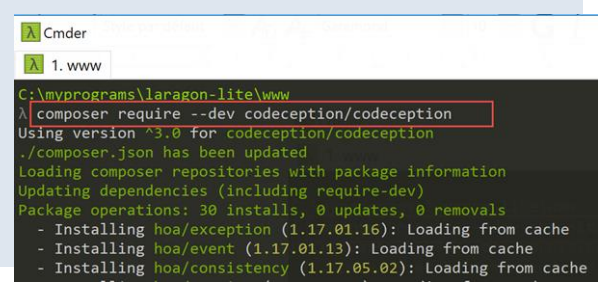
1.11.7.1 Installation du framework [Codeception]

Comme beaucoup de bibliothèques PHP, le framework **[Codeception]** s'installe avec **[Composer]**. Nous ouvrons donc un terminal Laragon (cf paragraphe lien).

Il nous faut tout d'abord installer le framework de tests PHPUnit **[https://phpunit.de/]**. En effet Codeception utilise en sous-main le framework PHPUnit:

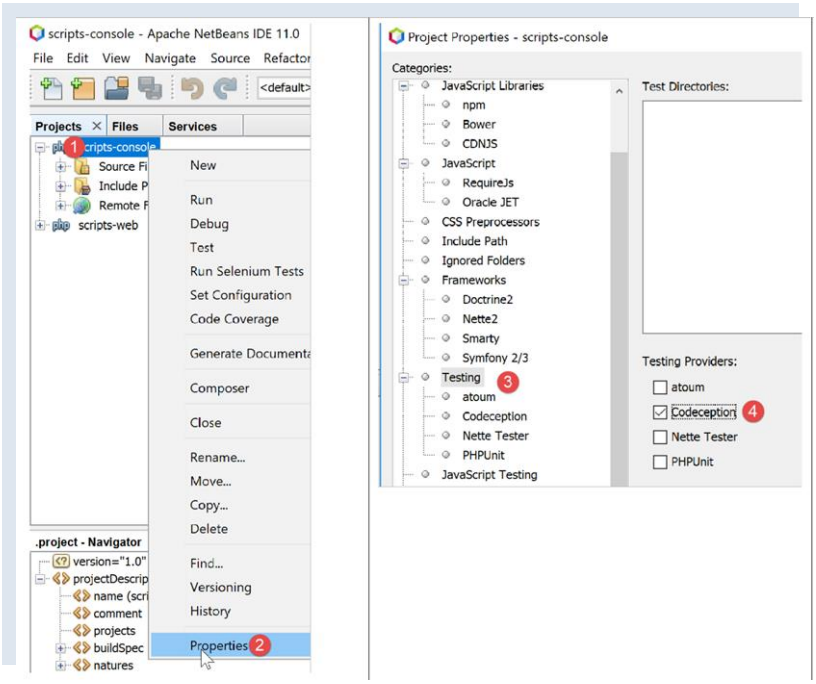


Ensuite, nous installons le framework Codeception :

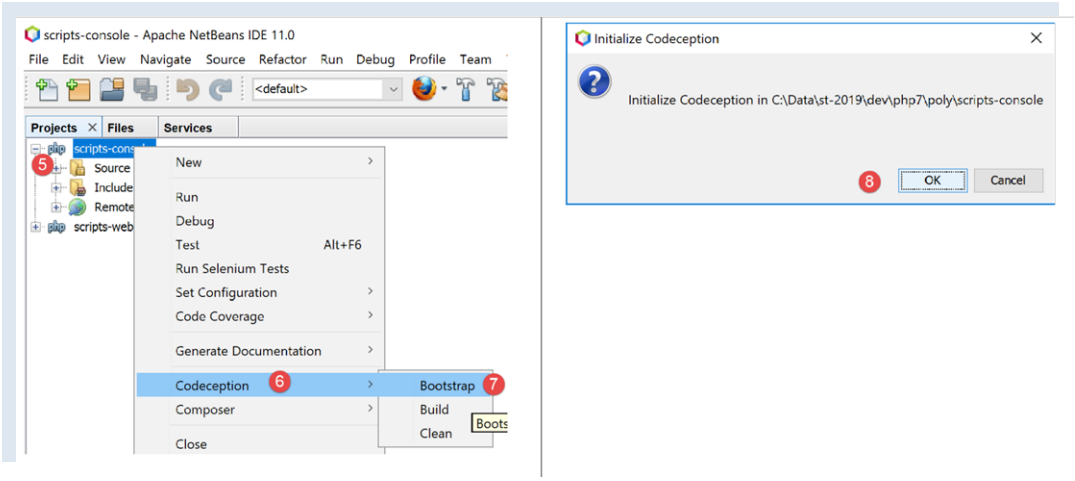


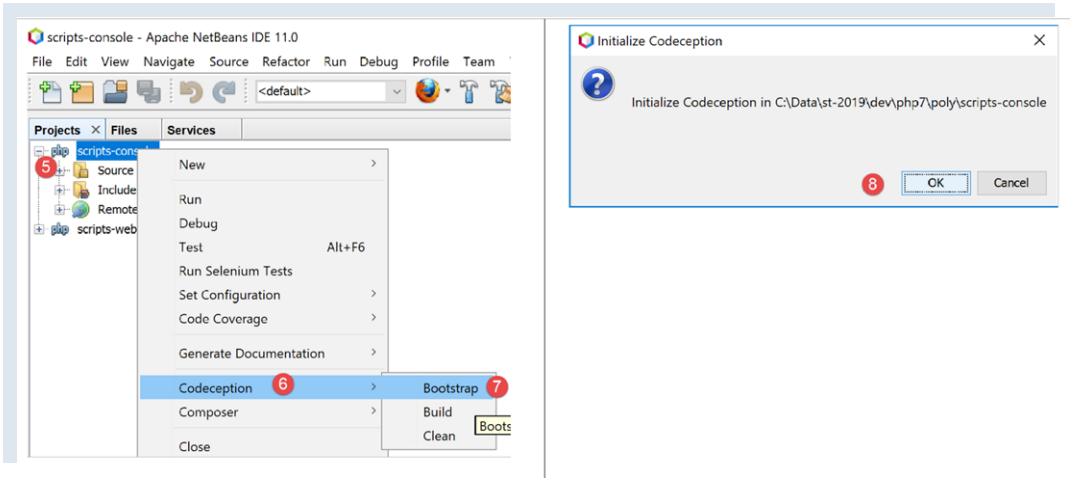
C'est tout. Maintenant voyons l'intégration de **[Codeception]** dans Netbeans.

1.11.7.2 Intégration de [CodeCeption] dans Netbeans

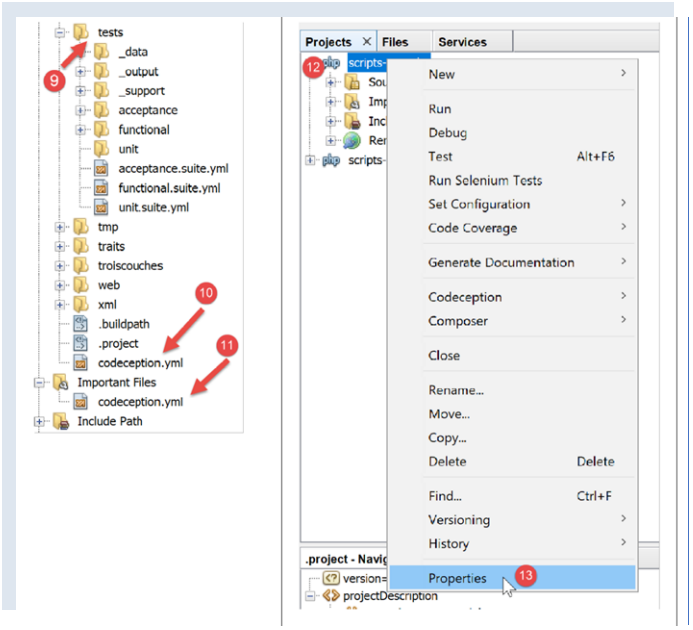


- en **[1-2]**, on accède aux propriétés du projet ;
- en **[3-4]**, on fait de **[Codeception]** l'un des frameworks de test du projet ;

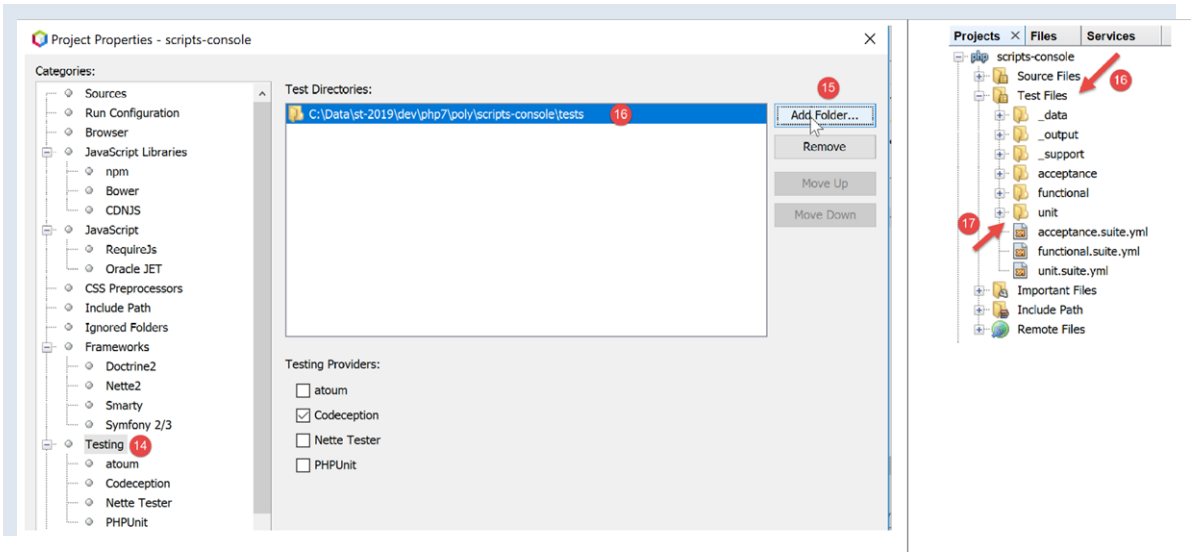




- en **[5-8]**, on initialise le framework **[Codeception]** pour le projet ;

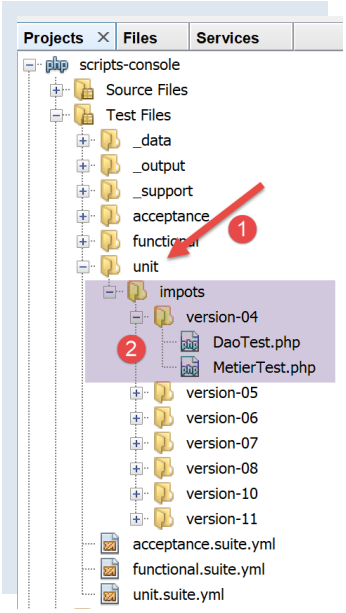


- en **[9]**, un dossier **[tests]** a été créé, ainsi qu'un fichier de configuration **[codeception.yml]** en **[10-11]**. Le fichier **[11]** est le même que le fichier **[10]**. Codeception a simplement créé un dossier **[Important Files]** pour donner une signification particulière au fichier **[10]** ;
- en **[12-13]**, on revient aux propriétés du projet ;



- en **[14-16]**, on désigne le dossier **[tests] [16]**, comme le dossier de tests du projet ;
- en **[16]**, le dossier **[tests]** apparaît alors sous le nouveau nom **[Test Files]**. La présence de ce dossier dans un projet PHP montre que ce projet intègre un framework de tests programmés ;
- nous créerons nos tests dans le dossier **[unit] [17] ;**

1.11.7.3 Tests de la couche [dao]



- nous allons créer tous nos tests dans le dossier **`[unit] [1]`** ;
- les noms des classes de test **`[Codeception]` doivent se terminer par le mot clé `[Test]`, sinon les classes ne seront pas reconnues comme classes de test** ;

Nos classes de test **`[Codeception]`** auront la forme suivante **`[https://codeception.com/docs/05-UnitTests]`** :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // chargement de l'environnement de test
10  …
11
12  class DaoTest extends \Codeception\Test\Unit {
13    // attributs du test
14    private $attribut1;
15
16    public function __construct() {
17      parent::__construct();
18      // initialisation de l'environnement de test
19      …
20    }
21
22    // tests
23    public function testTaxAdminData() {
24      // tests
25      $this->assertEquals($expected, $actual);
26      $this->assertEqualsWithDelta($expected, $actual, $delta);
27      $this->assertTrue($actual);
28      $this->assertFalse($actual);
29      $this->assertNull($actual);
30      $this->assertEmpty($actual);
31      $this→assertSame($expected, $actual);
32  …
33    }
34
35  }
```

Commentaires

- ligne 7 : les classes de test seront dans le même espace de noms que l'application testée ;
- lignes 9-10 : ici on trouvera les opérations **`[require]`** pour charger les classes et interfaces testées ;
- ligne 12 : le nom de la classe de test **doit obligatoirement se terminer par le mot clé `[Test]`**. Cette classe doit étendre la classe **`[\Codeception\Test\Unit]`** ;
- lignes 16-20 : le constructeur nous permettra d'initialiser l'environnement du test ;

- ligne 23 : les noms des méthodes de test **doivent obligatoirement commencer par le mot clé `[test]`** ;
- lignes 25-31 : diverses méthodes de test peuvent être utilisées ;

La classe de test **`[DaoTest]`** sera la suivante :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Application;
8.
9.  // constantes
10. define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-04");
11. define("VENDOR", "C:/myprograms/laragon-lite/www/vendor");
12. // inclusion interface et classes
13. require_once ROOT . "/TaxAdminData.php";
14. require_once ROOT . "/TaxPayerData.php";
15. require_once ROOT . "/ExceptionImpots.php";
16. require_once ROOT . "/Utilitaires.php";
17. require_once ROOT . "/InterfaceDao.php";
18. require_once ROOT . "/TraitDao.php";
19. require_once ROOT . "/DaoImpotsWithTaxAdminDataInJsonFile.php";
20. require_once ROOT . "/InterfaceMetier.php";
21. require_once ROOT . "/Metier.php";
22. require_once VENDOR. "/autoload.php";;
23. // test ------------------------------------------------------
24. // définition des constantes
25. const TAXADMINDATA_FILENAME = "taxadmindata.json";
26.
27. class DaoTest extends \Codeception\Test\Unit {
28.   // TaxAdminData
29.   private $taxAdminData;
30.
31.   public function __construct() {
32.     parent::__construct();
33.     // création de la couche [dao]
34.     $dao = new DaoImpotsWithTaxAdminDataInJsonFile(ROOT . "/" . TAXADMINDATA_FILENAME);
35.     $this->taxAdminData = $dao->getTaxAdminData();
36.   }
37.
38.   // tests
39.   public function testTaxAdminData() {
40.     …
41.   }
42.
43. }
```

Commentaires

Pour construire les tests d'une version de l'exercice d'application, nous utiliserons un environnement identique à celui utilisé par le script principal de la version. Celui de la version 04 est le script **`[main.php]`** suivant :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Application;
8.
9.  // gestion des erreurs par PHP
10. ini_set("display_errors", "0");
11.
12. // inclusion interface et classes
13. require_once __DIR__ . "/TaxAdminData.php";
14. require_once __DIR__ . "/TaxPayerData.php";
15. require_once __DIR__ . "/ExceptionImpots.php";
16. require_once __DIR__ . "/Utilitaires.php";
17. require_once __DIR__ . "/InterfaceDao.php";
18. require_once __DIR__ . "/TraitDao.php";
19. require_once __DIR__ . "/DaoImpotsWithTaxAdminDataInJsonFile.php";
20. require_once __DIR__ . "/InterfaceMetier.php";
21. require_once __DIR__ . "/Metier.php";
```

```
22. // test ------------------------------------------------------
23. // définition des constantes
24. const TAXPAYERSDATA_FILENAME = "taxpayersdata.txt";
25. const RESULTS_FILENAME = "resultats.txt";
26. const ERRORS_FILENAME = "errors.txt";
27. const TAXADMINDATA_FILENAME = "taxadmindata.json";
28.
29. try {
30.   // création de la couche [dao]
31.   $dao = new DaoImpotsWithTaxAdminDataInJsonFile(TAXADMINDATA_FILENAME);
32.   // création de la couche [métier]
33.   $métier = new Metier($dao);
34.   // calcul de l'impôts en mode batch
35.   $métier->executeBatchImpots(TAXPAYERSDATA_FILENAME, RESULTS_FILENAME, ERRORS_FILENAME);
36. } catch (ExceptionImpots $ex) {
37.   // on affiche l'erreur
38.   print $ex->getMessage() . "\n";
39. }
40. // fin
41. print "Terminé\n";
42. exit;
36
```

Pour tester la couche **[dao]**, dans la classe de test :

- nous reprenons l'environnement des lignes 13-27 de **[main.php]** ;
- dans le constructeur de la classe de test, nous construisons la couche **[dao]** comme en ligne 31 ;
- nous écrivons les méthodes de tests;

Nous procéderons de cette façon pour toutes les classes de test.

Revenons au code complet de la classe de test :

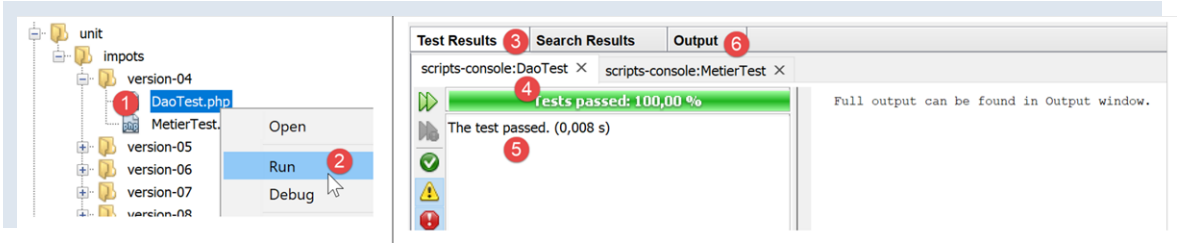
```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Application;
8.
9.  // constantes
10. define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-04");
11. define("VENDOR", "C:/myprograms/laragon-lite/www/vendor");
12. // inclusion interface et classes
13. require_once ROOT . "/TaxAdminData.php";
14. require_once ROOT . "/TaxPayerData.php";
15. require_once ROOT . "/ExceptionImpots.php";
16. require_once ROOT . "/Utilitaires.php";
17. require_once ROOT . "/InterfaceDao.php";
18. require_once ROOT . "/TraitDao.php";
19. require_once ROOT . "/DaoImpotsWithTaxAdminDataInJsonFile.php";
20. require_once ROOT . "/InterfaceMetier.php";
21. require_once ROOT . "/Metier.php";
22. require_once VENDOR. "/autoload.php";;
23. // test ------------------------------------------------------
24. // définition des constantes
25. const TAXADMINDATA_FILENAME = "taxadmindata.json";
26.
27. class DaoTest extends \Codeception\Test\Unit {
28.   // TaxAdminData
29.   private $taxAdminData;
30.
31.   public function __construct() {
32.     parent::__construct();
33.     // création de la couche [dao]
34.     $dao = new DaoImpotsWithTaxAdminDataInJsonFile(ROOT . "/" . TAXADMINDATA_FILENAME);
35.     $this->taxAdminData = $dao->getTaxAdminData();
36.   }
37.
38.   // tests
39.   public function testTaxAdminData() {
40.     // constantes de calcul
41.     $this->assertEquals(1551, $this->taxAdminData->getPlafondQfDemiPart());
42.     $this->assertEquals(21037, $this->taxAdminData->getPlafondRevenusCelibataire PourReduction());
```

```
43.     $this->assertEquals(42074, $this->taxAdminData->getPlafondRevenusCouplePourReduction());
44.     $this->assertEquals(3797, $this->taxAdminData->getValeurReducDemiPart());
45.     $this->assertEquals(1196, $this->taxAdminData->getPlafondDecoteCelibataire());
46.     $this->assertEquals(1970, $this->taxAdminData->getPlafondDecoteCouple());
47.     $this->assertEquals(1595, $this->taxAdminData->getPlafondImpotCelibatairePourDecote());
48.     $this->assertEquals(2627, $this->taxAdminData->getPlafondImpotCouplePourDecote());
49.     $this->assertEquals(12502, $this->taxAdminData->getAbattementDixPourcentMax());
50.     $this->assertEquals(437, $this->taxAdminData->getAbattementDixPourcentMin());
51.     // tranches de l'impôt
52.     $this->assertSame([9964.0, 27519.0, 73779.0, 156244.0, 0.0], $this->taxAdminData->getLimites());
53.     $this->assertSame([0.0, 0.14, 0.30, 0.41, 0.45], $this->taxAdminData->getCoeffR());
54.     $this->assertSame([0.0, 1394.96, 5798.0, 13913.69, 20163.45], $this->taxAdminData->getCoeffN());
55.   }
56.
57. }
```

Commentaires

- lignes 10-25 : chargement de l'environnement nécessaire aux tests et définitions de constantes ;
- lignes 31-36 : construction de la couche **[dao]**, ligne 34, puis initialisation de l'attribut **[$taxAdminData]** de la ligne 29. Cet attribut contient les données de l'administration fiscale ;
- lignes 39-55 : l'unique méthode de test. Celui-ci consiste à vérifier que le contenu de l'attribut **[$taxAdminData]** correspond à ce qui est attendu ;
- lignes 41-50 : vérifications des constantes du calcul de l'impôt ;
- lignes 52-55 : vérifications des tranches d'imposition. La méthode **[assertSame]** vérifie que deux entités PHP, ici des tableaux, sont identiques ;

Pour exécuter cette classe de test, on procède de la façon suivante :



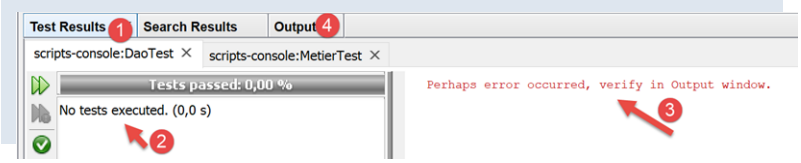
- en **[1-2]**, on exécute le test ;
- **[3]** : la fenêtre des résultats des tests ;
- **[4]** : la classe de test exécutée ;
- **[5]** : les résultats. Ici l'unique méthode de test a été réussie ;
- **[6]** : lorsque le test échoue ou plus fréquemment lorsqu'aucun test n'a été exécuté, il faut aller voir la fenêtre **[6]**. Le plus souvent, c'est le chargement de l'environnement du test qui a échoué et aucun test n'a alors pu être exécuté. Les erreurs affichées dans **[6]** sont celles qu'on aurait avec l'exécution d'un script PHP classique ;

Montrons un exemple de test erroné :

Dans la classe de test, nous introduisons une erreur dans la définition d'une constante :

```
// constantes
define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-04x");
```

puis nous exécutons le test. Le résultat obtenu est le suivant :



Dans la fenêtre **[4]** :

1.11.7.4 Tests de la couche [métier]

La classe de test **[MetierTest]** suit les mêmes règles de construction que la classe **[DaoTest]** mais il y a plus de méthodes de test :

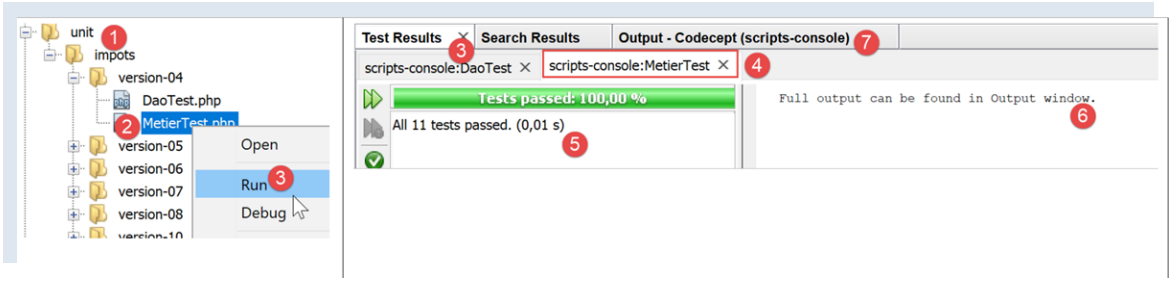
```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Application;
8.
9.  // constantes
10. define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-04");
11. define("VENDOR", "C:/myprograms/laragon-lite/www/vendor");
12. // inclusion interface et classes
13. require_once ROOT . "/TaxAdminData.php";
14. require_once ROOT . "/TaxPayerData.php";
15. require_once ROOT . "/ExceptionImpots.php";
16. require_once ROOT . "/Utilitaires.php";
17. require_once ROOT . "/InterfaceDao.php";
18. require_once ROOT . "/TraitDao.php";
19. require_once ROOT . "/DaoImpotsWithTaxAdminDataInJsonFile.php";
20. require_once ROOT . "/InterfaceMetier.php";
21. require_once ROOT . "/Metier.php";
22. require_once VENDOR. "/autoload.php";;
23. // test --------------------------------------------------
24. // définition des constantes
25. const TAXADMINDATA_FILENAME = "taxadmindata.json";
26.
27. class MetierTest extends \Codeception\Test\Unit {
28.   // couche métier
29.   private $métier;
30.
31.   public function __construct() {
32.     parent::__construct();
33.     // création de la couche [dao]
34.     $dao = new DaoImpotsWithTaxAdminDataInJsonFile(ROOT . "/" . TAXADMINDATA_FILENAME);
35.     // création de la couche [métier]
36.     $this->métier = new Metier($dao);
37.   }
38.
39.   // tests
40.   public function test1() {
41.     $result = $this->métier->calculerImpot("oui", 2, 55555);
42.     $this->assertEqualsWithDelta(2815, $result["impôt"], 1);
43.     $this->assertEqualsWithDelta(0, $result["surcôte"], 1);
44.     $this->assertEqualsWithDelta(0, $result["décôte"], 1);
45.     $this->assertEqualsWithDelta(0, $result["réduction"], 1);
46.     $this->assertEquals(0.14, $result["taux"]);
47.   }
48.
49.   public function test2() {
50.     $result = $this->métier->calculerImpot("oui", 2, 50000);
51.     $this->assertEqualsWithDelta(1385, $result["impôt"], 1);
52.     $this->assertEqualsWithDelta(0, $result["surcôte"], 1);
53.     $this->assertEqualsWithDelta(384, $result["décôte"], 1);
54.     $this->assertEqualsWithDelta(347, $result["réduction"], 1);
55.     $this->assertEquals(0.14, $result["taux"]);
56.   }
57.
58.   public function test3() {
59.     $result = $this->métier->calculerImpot("oui", 3, 50000);
60.     $this->assertEqualsWithDelta(0, $result["impôt"], 1);
61.     $this->assertEqualsWithDelta(0, $result["surcôte"], 1);
```

```
62.     $this->assertEqualsWithDelta(720, $result["décôte"], 1);
63.     $this->assertEqualsWithDelta(0, $result["réduction"], 1);
64.     $this->assertEquals(0.14, $result["taux"]);
65.   }
66.
67.   public function test4() {
68.     $result = $this->métier->calculerImpot("non", 2, 100000);
69.     $this->assertEqualsWithDelta(19884, $result["impôt"], 1);
70.     $this->assertEqualsWithDelta(4480, $result["surcôte"], 1);
71.     $this->assertEqualsWithDelta(0, $result["décôte"], 1);
72.     $this->assertEqualsWithDelta(0, $result["réduction"], 1);
73.     $this->assertEquals(0.41, $result["taux"]);
74.   }
75.
76.   public function test5() {
77.     $result = $this->métier->calculerImpot("non", 3, 100000);
78.     $this->assertEqualsWithDelta(16782, $result["impôt"], 1);
79.     $this->assertEqualsWithDelta(7176, $result["surcôte"], 1);
80.     $this->assertEqualsWithDelta(0, $result["décôte"], 1);
81.     $this->assertEqualsWithDelta(0, $result["réduction"], 1);
82.     $this->assertEquals(0.41, $result["taux"]);
83.   }
84.
85.   public function test6() {
86.     $result = $this->métier->calculerImpot("oui", 3, 100000);
87.     $this->assertEqualsWithDelta(9200, $result["impôt"], 1);
88.     $this->assertEqualsWithDelta(2180, $result["surcôte"], 1);
89.     $this->assertEqualsWithDelta(0, $result["décôte"], 1);
90.     $this->assertEqualsWithDelta(0, $result["réduction"], 1);
91.     $this->assertEquals(0.3, $result["taux"]);
92.   }
93.
94.   public function test7() {
95.     $result = $this->métier->calculerImpot("oui", 5, 100000);
96.     $this->assertEqualsWithDelta(4230, $result["impôt"], 1);
97.     $this->assertEqualsWithDelta(0, $result["surcôte"], 1);
98.     $this->assertEqualsWithDelta(0, $result["décôte"], 1);
99.     $this->assertEqualsWithDelta(0, $result["réduction"], 1);
100.    $this->assertEquals(0.14, $result["taux"]);
101.  }
102.
103.  public function test8() {
104.    $result = $this->métier->calculerImpot("non", 0, 100000);
105.    $this->assertEqualsWithDelta(22986, $result["impôt"], 1);
106.    $this->assertEqualsWithDelta(0, $result["surcôte"], 1);
107.    $this->assertEqualsWithDelta(0, $result["décôte"], 1);
108.    $this->assertEqualsWithDelta(0, $result["réduction"], 1);
109.    $this->assertEquals(0.41, $result["taux"]);
110.  }
111.
112.  public function test9() {
113.    $result = $this->métier->calculerImpot("oui", 2, 30000);
114.    $this->assertEqualsWithDelta(0, $result["impôt"], 1);
115.    $this->assertEqualsWithDelta(0, $result["surcôte"], 1);
116.    $this->assertEqualsWithDelta(0, $result["décôte"], 1);
117.    $this->assertEqualsWithDelta(0, $result["réduction"], 1);
118.    $this->assertEquals(0, $result["taux"]);
119.  }
120.
121.  public function test10() {
122.    $result = $this->métier->calculerImpot("non", 0, 200000);
123.    $this->assertEqualsWithDelta(64210, $result["impôt"], 1);
124.    $this->assertEqualsWithDelta(7498, $result["surcôte"], 1);
125.    $this->assertEqualsWithDelta(0, $result["décôte"], 1);
126.    $this->assertEqualsWithDelta(0, $result["réduction"], 1);
127.    $this->assertEquals(0.45, $result["taux"]);
128.  }
129.
130.  public function test11() {
131.    $result = $this->métier->calculerImpot("oui", 3, 200000);
132.    $this->assertEqualsWithDelta(42842, $result["impôt"], 1);
133.    $this->assertEqualsWithDelta(17283, $result["surcôte"], 1);
134.    $this->assertEqualsWithDelta(0, $result["décôte"], 1);
135.    $this->assertEqualsWithDelta(0, $result["réduction"], 1);
136.    $this->assertEquals(0.41, $result["taux"]);
137.  }
138.}
```

Commentaires

- lignes 10-25 : chargements des fichiers définissant l'environnement du test. Celui-ci est le même que pour la couche **[dao]** ;
- lignes 31-37 : instanciation des couches **[dao]** et **[métier]** ;
- lignes 40-47 : un test de calcul d'impôt ;
- ligne 41 : un certain calcul d'impôt est fait avec la couche **[métier]** ;
- lignes 42-46 : on vérifie que les résultats obtenus sont ceux du simulateur de l'administration fiscale **[https://www3.impots.gouv.fr/simulateur/calcul_impot/2019/simplifie/index.htm]** ;
- lignes 23-26 : les tests d'égalité sont faits à 1 euro près. En effet, on a vu que des problèmes d'arrondi faisaient que l'algorithme du document donnait les résultats attendus à 1 euro près ;
- ligne 27 : le taux d'imposition est lui calculé sans marge d'erreur ;
- lignes 49-137 : on répète ce type de tests 10 fois avec à chaque fois une configuration du contribuable différente ;

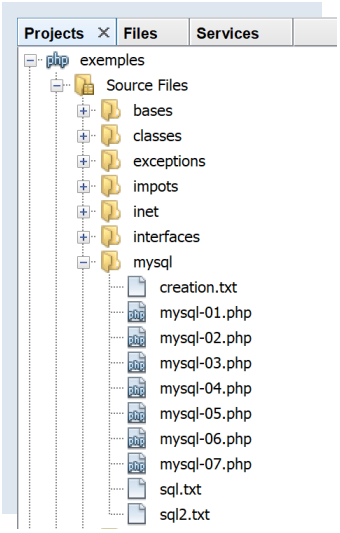
Les tests donnent les résultats suivants :



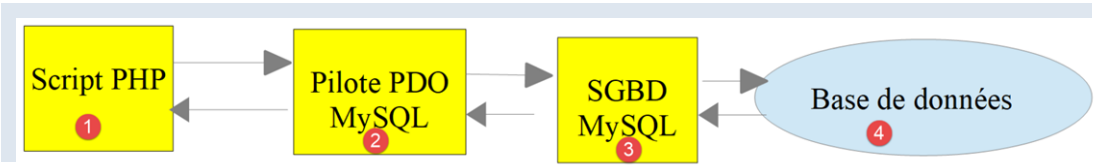
1.11.7.5 Tests des prochaines versions

Dans la suite, les tests des couches **[dao]** et **[métier]** seront identiques à ceux de la version 04. Seul changera l'environnement du test. Nous ne présenterons donc que celui-ci et les résultats des tests.

1.12 Utilisation du SGBD MySQL

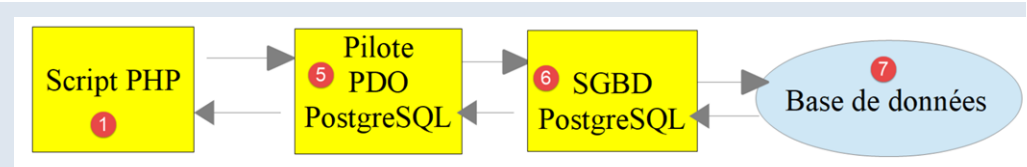


Nous allons maintenant écrire des scripts PHP utilisant une base de données MySQL :



Dans l'architecture ci-dessus, le script PHP (1) ne dialogue pas directement avec le SGBD (**S**ystème de **G**estion de **B**ases de **D**onnées) (3). Il dialogue avec un intermédiaire appelé **pilote** de SGBD ou encore **driver** de SGBD. PHP fournit une interface standard pour

ces pilotes, l'interface **PDO** (**P**HP **D**ata **O**bjects). Cette interface est implémentée par différentes classes adaptées à chaque SGBD : une classe pour le SGBD MySQL, une autre pour le SGBD PostgreSQL… Pour changer de SGBD, on change de pilote :



Le pilote PDO isole le script PHP (1) du SGBD (3, 6). Comme ces pilotes implémentent une interface standard, on peut s'attendre à ce que le script PHP (1) ne change pas si on passe du SGBD MySQL (3) au SGBD PostgreSQL (6). Dans la réalité cet idéal n'existe pas. En effet pour dialoguer avec le SGBD, le script PHP envoie des ordres SQL (**S**tandard **Q**uery **L**anguage). C'est un langage implémenté par tous les SGBD mais qui est incomplet. Aussi les SGBD lui ont-ils ajouté des ordres propriétaires. C'est une première cause d'incompatibilité entre SGBD. Par ailleurs les types de données utilisables dans les bases de données peuvent être différents d'un SGBD à l'autre. Ainsi PostgreSQL accepte un nombre de types de données bien plus grand que le SGBD MySQL. C'est une seconde cause d'incompatibilité. Une autre cause est la gestion des clés primaires automatiques (générées par le SGBD) : quasiment chaque SGBD a sa propre politique. Etc… Les causes d'incompatibilité sont nombreuses.

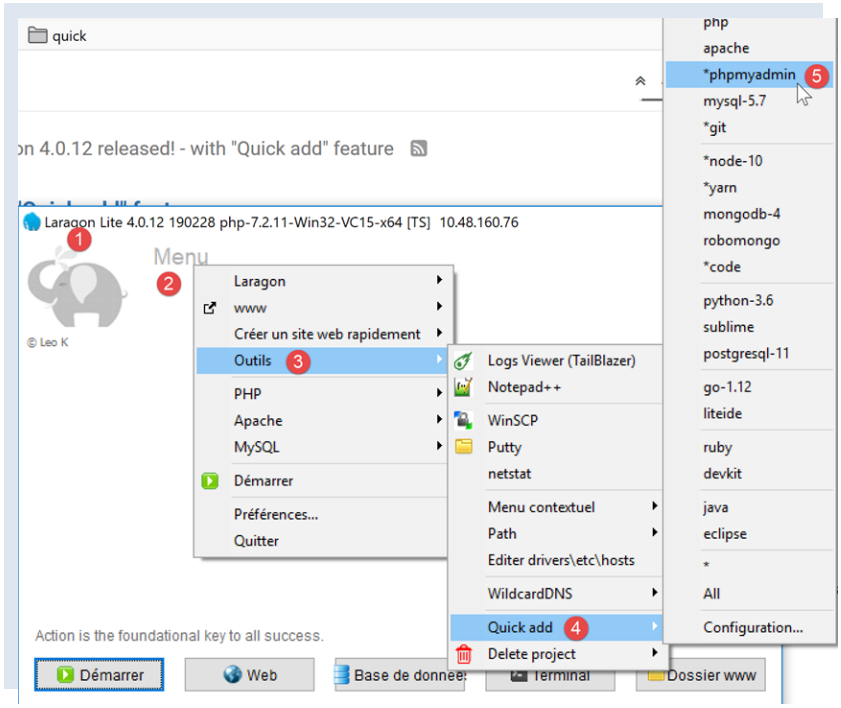
Si on veut éviter de réécrire le script PHP (1) en passant de MySQL (3) à PostgreSQL (6), on est généralement amenés à insérer une nouvelle couche entre le script PHP (1) et le pilote PDO (2, 5) dont le rôle sera de gommer les incompatibilités entre les deux SGBD. Cependant dans les cas simples que nous allons rencontrer, cette couche supplémentaire ne sera pas nécessaire.

Nous allons maintenant utiliser le SGBD MySQL. Celui-ci est inclus dans le paquetage Laragon (cf paragraphe lien).

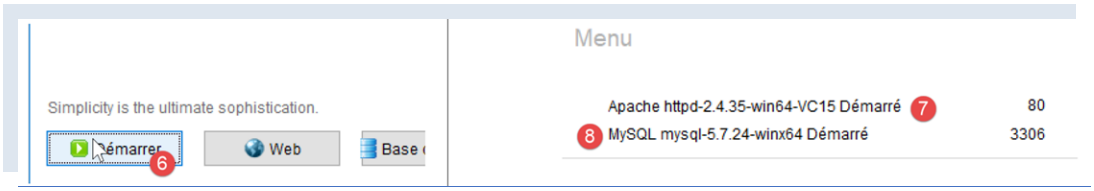
Si le lecteur est novice avec la notion de base de données et de langage SQL, il pourra lire le document **[http://sergetahe.com/cours-tutoriels-de-programmation/cours-tutoriel-sql-avec-le-sgbd-firebird/]**. Ce document utilise le SGBD Firebird et non pas MySQL mais donne les fondamentaux des bases de données et du langage SQL. Comme MySQL, Firebird dispose d'une version librement utilisable et de faible empreinte mémoire.

1.12.1 Création d'une base de données

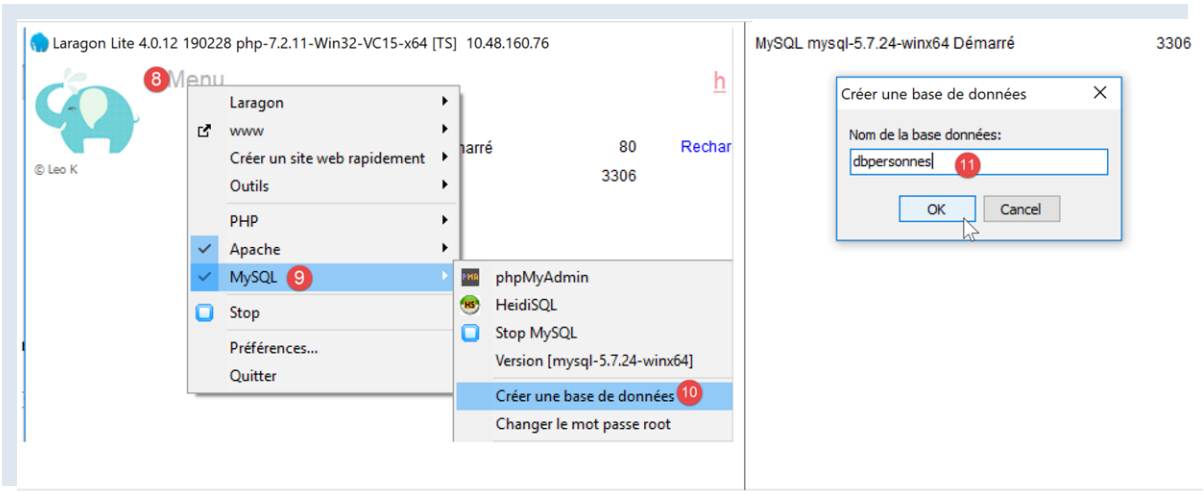
Nous montrons maintenant comment créer une base de données ainsi qu'un utilisateur MySQL avec l'outil Laragon.



- une fois lancé, Laragon **[1]** peut être administré à partir d'un menu **[2]** ;
- en **[3-5]**, on installe l'outil **[phpMyAdmin]** d'administration de MySQL s'il n'a pas déjà été installé ;

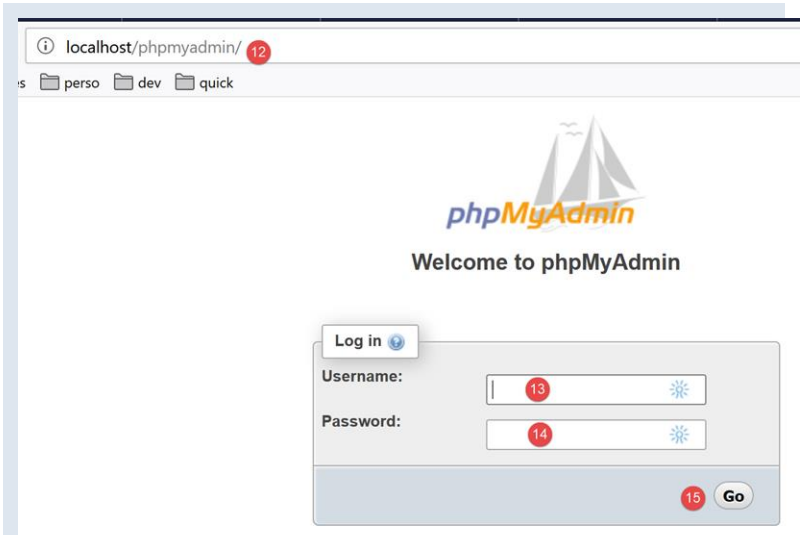


- en **[6]**, on démarre le serveur web Apache ainsi que le SGBD MySQL ;
- en **[7]**, le serveur Apache est lancé ;
- en **[8]**, le SBD MySQL est lancé ;



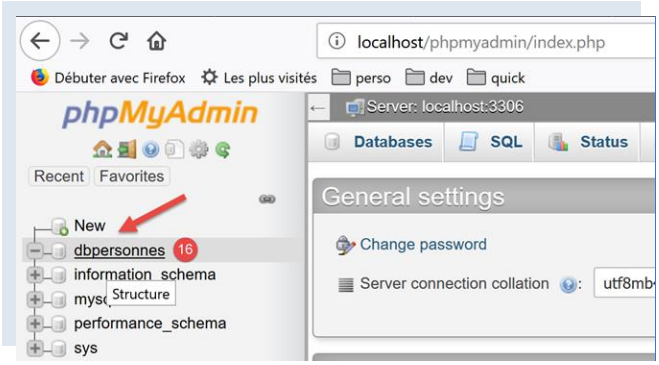
- en **[8-10]**, on crée une base de données qu'on nomme **[dbpersonnes] [11]**. On va construire une base de données de personnes ;

- en **[11]**, on va gérer la base de données qu'on vient de créer ;

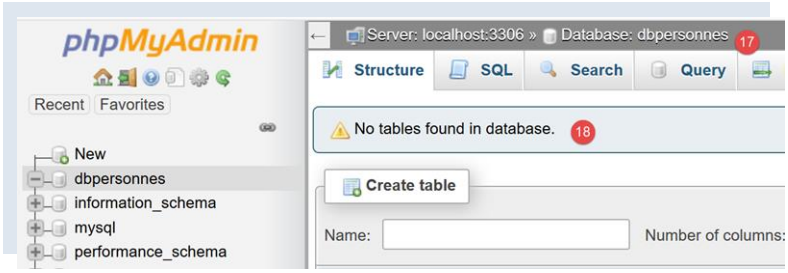


- l'opération **[Bases de données]** émet une requête web vers l'URL **[http://localhost/phpmyadmin]**. C'est le serveur web Apache de Laragon qui répond. L'URL **[http://localhost/phpmyadmin]** est l'URL de l'utilitaire **[phpMyAdmin]** que nous avons installé précédemment **[5]**. Cet utilitaire permet de gérer les bases de données MySQL ;

- par défaut, les identifiants de connexion de l'administrateur de la base sont : **root `[13]`** sans mot de passe **`[14]`** ;

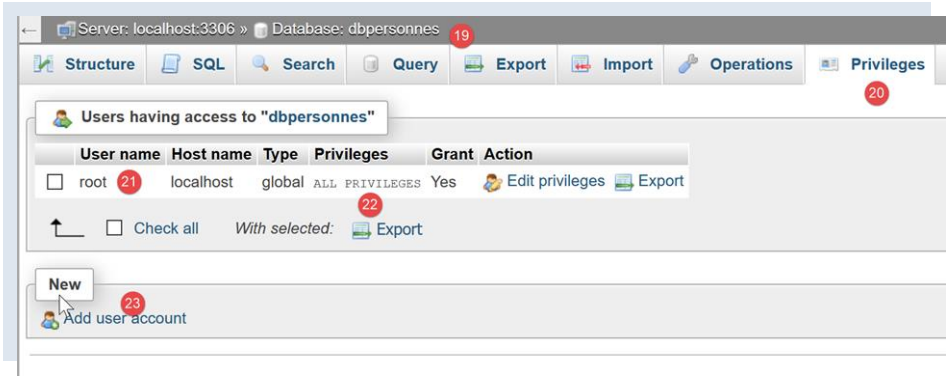


- en **`[16]`**, la base de données que nous avons créée précédemment ;

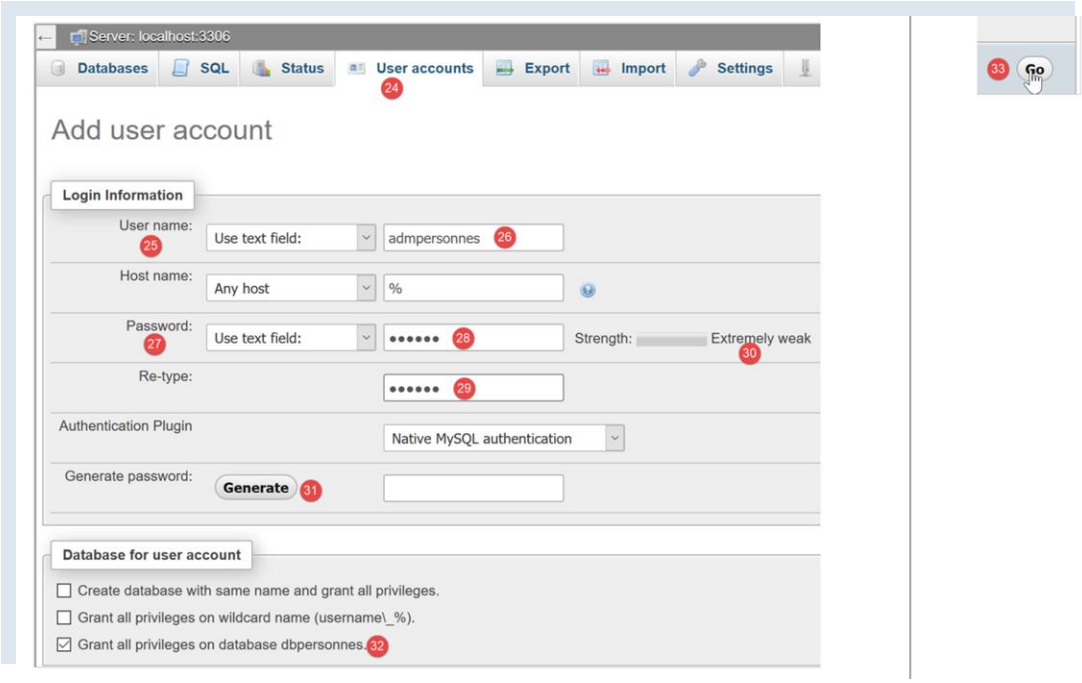


- on a pour l'instant une base **`[dbpersonnes]`** **`[17]`** qui est vide **`[18]`** ;

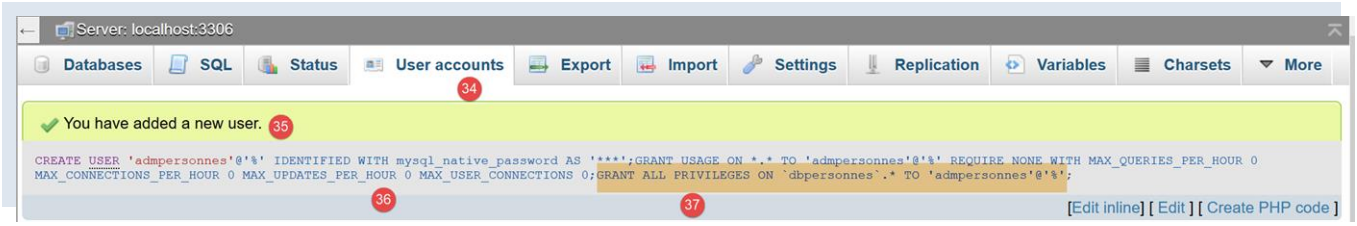
On crée un utilisateur **`[admpersonnes]`** avec le mot de passe **`[nobody]`** qui va avoir tous les droits sur la base de données **`[dbpersonnes]`** :



- en **`[19]`**, on est positionnés sur la base **`[dbpersonnes]`** ;
- en **`[20]`**, on sélectionne l'onglet **`[Privileges]`** ;
- en **`[21-22]`**, on voit que l'utilisateur **`[root]`** a tous les droits sur la base **`[dbpersonnes]`** ;
- en **`[23]`**, on crée un nouvel utilisateur ;



- en **[25-26]**, l'utilisateur aura l'identifiant **[admdbpersonnes]** ;
- en **[27-29]**, son mot de passe sera **[nobody]** ;
- en **[30]**, phpMyAdmin signale que le mot de passe est très faible (facile à craquer). En production, il est préférable de générer un mot de passe fort avec **[31]** ;
- en **[32]**, on indique que l'utilisateur **[admdbpersonnes]** doit avoir tous les droits sur la base **[dbpersonnes]** ;
- en **[33]**, on valide les renseignements donnés ;



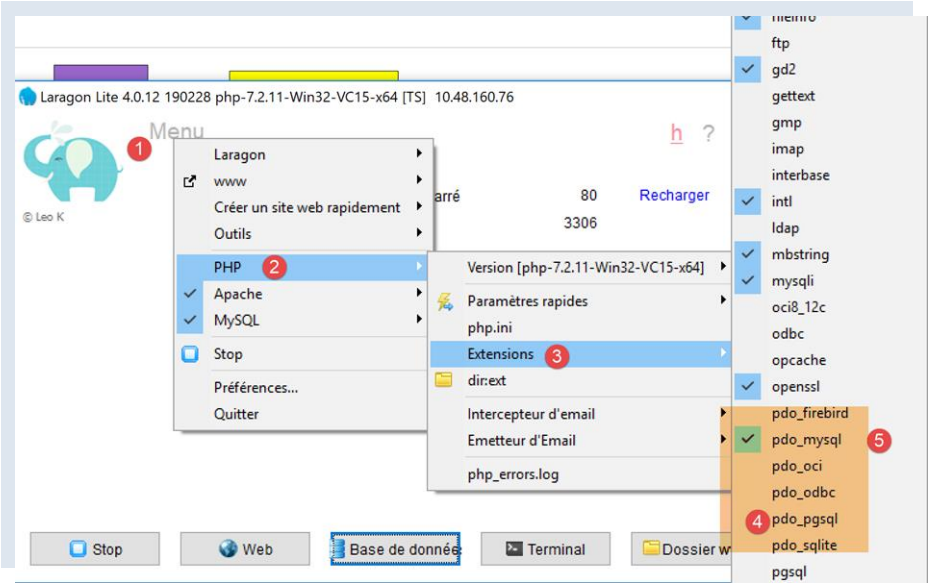
- en **[35]**, phpMyAdmin indique que l'utilisateur a été créé ;
- en **[36]**, l'ordre SQL qui a été émis sur la base ;
- en **[37]**, l'utilisateur **[admpersonnes]** a tous les droits sur la base de données **[dbpersonnes]** ;

Désormais nous avons :

- une base de données MySQL **[dbpersonnes]** ;
- un utilisateur **[admpersonnes/nobody]** qui a tous les droits sur cette base de données ;

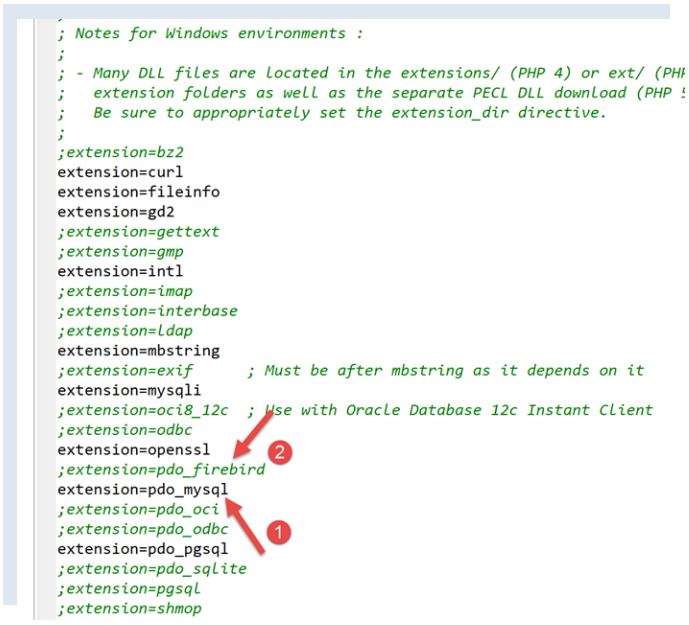
Nous allons écrire des scripts PHP pour exploiter la base de données. PHP dispose de diverses bibliothèques pour gérer les bases de données. Nous utiliserons la bibliothèque PDO (PHP Data Objects) qui s'intercale entre le code PHP et le SGBD :

La bibliothèque PDO permet au script PHP de s'abstraire de la nature exacte du SGBD utilisé. Ainsi ci-dessus, le SGBD **MySQL** peut être remplacé par le SGBD **PostgreSQL** avec un impact minimum sur le code du script PHP. Cette bibliothèque n'est pas disponible par défaut. On peut vérifier sa disponibilité de la façon suivante :



- en **[1-4]**, on vérifie les extensions PDO actives ;
- en **[5]**, on voit que l'extension PDO pour le SGBD MySQL est active. Les autres ne le sont pas. Il suffirait de les cliquer pour les activer ;

Une autre façon d'activer une extension est de modifier directement le fichier **[php.ini]** (paragraphe lien) qui configure PHP :



- en **[1]**, l'extension PDO de MySQL est activée ;
- en **[2]**, l'extension PDO de Firebird est désactivée ;

Après avoir modifié le fichier **[php.ini]**, il faut **relancer** le PHP de Laragon pour que les modifications soient prises en compte.

1.12.2 Connexion à une base de données MySQL

La connexion à un SGBD se fait par la construction d'un objet PDO. Le constructeur admet différents paramètres :

```
$dbh=new PDO(string $dsn,string $user,string $passwd,array $driver_options)
```

La signification des paramètres est la suivante :

$dsn	(Data Source Name) est une chaîne précisant la nature du SGBD et sa localisation sur internet. La chaîne "*mysql:host=localhost*" indique qu'on a affaire à un SGBD MySQL opérant sur le serveur local. Cette chaîne peut comprendre d'autres paramètres, notamment le port d'écoute du SGBD et le nom de la base à laquelle on veut se connecter : "*mysql:host=localhost;port=3306;dbname=dbpersonnes*" ;
$user	identifiant de l'utilisateur qui se connecte ;
$passwd	son mot de passe ;
$driver_options	un tableau d'options pour le pilote du SGBD ;

Seul le premier paramètre est obligatoire. L'objet ainsi construit sera ensuite le support de toutes les opérations faites sur la base de données à laquelle on s'est connecté. Si l'objet PDO n'a pu être construit, une exception de type **PDOException** est lancée.

Voici un exemple de connexion **[mysql-01.php]** :

```
1.  <?php
2.
3.  // connexion à une base MySql locale
4.  // l'identité de l'utilisateur est (admpersonnes,nobody)
5.  const ID = "admpersonnes";
6.  const PWD = "nobody2";
7.  const HOTE = "localhost";
8.
9.  try {
10.   // connexion
11.   $dbh = new PDO("mysql:host=".HOTE, ID, PWD);
12.   print "Connexion réussie\n";
13.   // fermeture de la connexion
14.   $dbh = NULL;
15. } catch (PDOException $e) {
16.   print "Erreur : " . $e->getMessage() . "\n";
17.   exit();
18. }
```

Résultats :

```
1   Connexion réussie
```

Commentaires

- ligne 11 : la connexion à un SGBD se fait par la construction d'un objet PDO. Le constructeur est ici utilisé avec les paramètres suivants :
  - une chaîne précisant la nature du SGBD et sa localisation sur internet. La chaîne "*mysql:host=localhost*" indique qu'on a affaire à un SGBD MySQL opérant sur le serveur local. Le port n'a pas été précisé. Le port 3306 est alors utilisé par défaut. Le nom de la base de données n'est pas indiqué non plus. Il y aura alors connexion au SGBD MySQL, la sélection d'une base précise se faisant plus tard ;
  - un identifiant d'utilisateur ;
  - son mot de passe ;
- ligne 14 : la fermeture de la connexion se fait par suppression de l'objet PDO créé initialement ;
- ligne 15 : la connexion à un SGBD peut échouer. Dans ce cas, une exception de type PDOException est lancée. Celle-ci dérive de l'exception PHP **[RuntimeException]** ;
- ligne 16 : on affiche le message d'erreur de l'exception ;

Réexécutons le script en mettant ligne 6 un mot de passe erroné. Le résultat est alors le suivant :

```
Erreur : SQLSTATE[HY000] [1045] Access denied for user 'admpersonnes'@'localhost' (using password: YES)
```

1.12.3 Création d'une table

Le script **[mysql-02.php]** montre la création d'une table dans une base de données :

```
1.  <?php
2.
3.  // identité de la base de données
4.  const DSN = "mysql:host=localhost;dbname=dbpersonnes";
5.  // identifiants de l'utilisateur
6.  const ID = "admpersonnes";
7.  const PWD = "nobody";
8.
9.  try {
10.   // connexion à la base MySql
11.   $connexion = new PDO(DSN, ID, PWD);
```

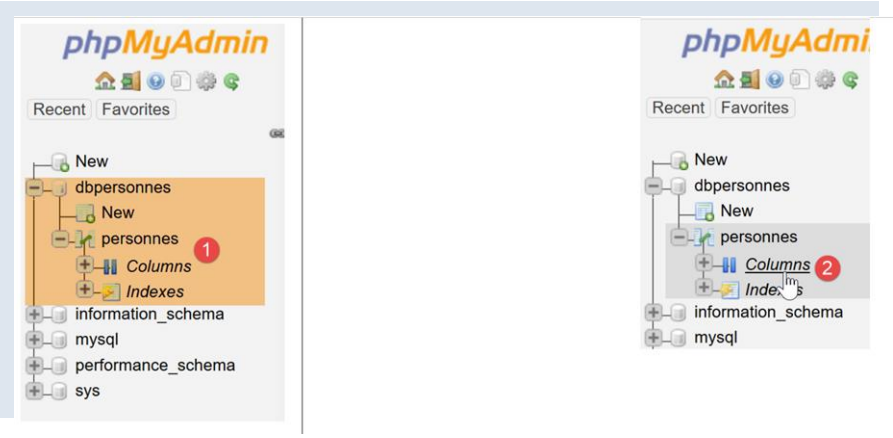
```
12.   // suppression de la table personnes si elle existe
13.   $sql = "drop table personnes";
14.   $connexion->exec($sql);
15.   // création de la table personnes
16.   $sql = "create table personnes (prenom varchar(30) NOT NULL, nom varchar(30) NOT NULL, age integer NOT NU
    LL, primary key(nom,prenom))";
17.   $connexion->exec($sql);
18. } catch (PDOException $ex) {
19.   // affichage erreur
20.   print "Erreur : " . $ex->getMessage() . "\n";
21. } finally {
22.   // on se déconnecte si besoin est
23.   $connexion = NULL;
24. }
25. // fin
26. print "Terminé\n";
27. exit;
```

Commentaires

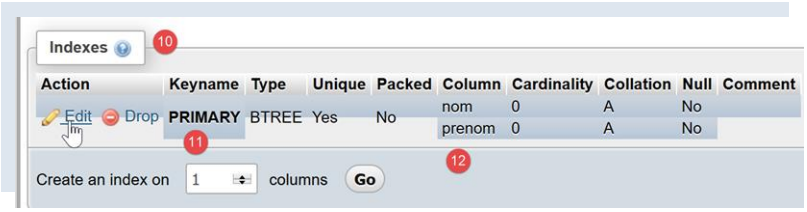
- ligne 11 : connexion à la base de données. C'est toujours la 1re chose à faire. Le résultat de la connexion est un objet **[PDO]** au travers duquel vont prendre les opérations sur la base de données ;
- ligne 13 : l'ordre SQL **[drop table personnes]** va supprimer la table **[personnes]** de la base de données **[dbpersonnes]**. Si la table **[personnes]** n'existe pas, cela ne provoque pas d'erreur ;
- ligne 14 : exécution de l'ordre SQL précédent sur la base de données **[dbpersonnes]**. Cet exécution peut lancer une **[PDOException]** qui sera interceptée ligne 18 ;
- ligne 16 : cet ordre SQL crée une table **[personnes]**. Une table contient des lignes et des colonnes. Les colonnes forment ce qu'on appelle la **structure** de la table. Les lignes forment le **contenu** de la table. Une base de données peut contenir une ou plusieurs tables. La table **[personnes]** aura ici trois colonnes :
  - **prenom** : le prénom d'une personne sous la forme d'une chaîne d'au plus 30 caractères ;
  - **nom** : le nom de cette même personne sous la forme d'une chaîne d'au plus 30 caractères ;
  - **age** : l'âge de la personne sous la forme d'un entier ;
  - l'attribut **NOT NULL** sur une colonne impose que la colonne ait une valeur. Ne pas lui en donner provoque une **[PDOException]** ;
  - **[primary key(nom,prenom)]** fixe une **clé primaire** à la table **[personnes]**. Une clé primaire a une valeur **unique** pour chaque ligne de la table. Ici la clé primaire sera obtenue par concaténation des colonnes **[nom]** et **[prenom]** de la ligne. Cette contrainte fait que dans la table on ne pourra pas avoir deux personnes ayant les mêmes nom et prénom, donc deux homonymes. Créer un homonyme d'une personne dans la table provoque une **[PDOException]** ;
- ligne 17 : exécution de l'ordre SQL sur la base de données **[dbpersonnes]** ;
- ligne 20 : s'il se produit une **[PDOException]**, on affiche le message d'erreur associé ;
- lignes 21-24 : on passe dans la clause **[finally]** dans tous les cas, exception ou pas, pour fermer la connexion à la base de données (ligne 23) ;

Résultats :

Si l'exécution du script se passe sans erreurs, on peut voir la présence de la table dans phpMyAdmin :



- en **[3]** la base de données ;
- en **[4]**, la table présentée ;
- en **[5]**, la structure des tables est présentée dans l'onglet **[Structure]** ;
- en **[6-8]**, les trois colonnes de la table ;
- en **[9]**, aucune des trois colonnes ne peut être vide ;



- en **[10]**, la liste des index de la table. Un index permet de retrouver dans la table les lignes ayant tel index, plus vite que si on parcourait séquentiellement les lignes de la table. La clé primaire fait toujours partie des index mais un index peut ne pas être une clé primaire ;
- en **[11]**, l'index est ici la clé primaire ;
- en **[12]**, l'index est constitué des colonnes **[nom, prenom]** de chaque ligne ;

Maintenant, voyons ce qui se passe si on crée des erreurs, respectivement sur le nom de la base, le nom de l'utilisateur, son mot de passe :
Si on met un nom de base inexistante :

```
Erreur : SQLSTATE[HY000] [1044] Access denied for user 'admpersonnes'@'%' to database 'dbpersonnes2'
```

Si on met un nom d'utilisateur inexistant :

```
Erreur : SQLSTATE[HY000] [1045] Access denied for user 'admpersonnes2'@'localhost' (using password:
YES)
```

Si on met un mot de passe erroné :

```
Erreur : SQLSTATE[HY000] [1045] Access denied for user 'admpersonnes'@'localhost' (using password: YES)
```

1.12.4 Remplissage d'une table

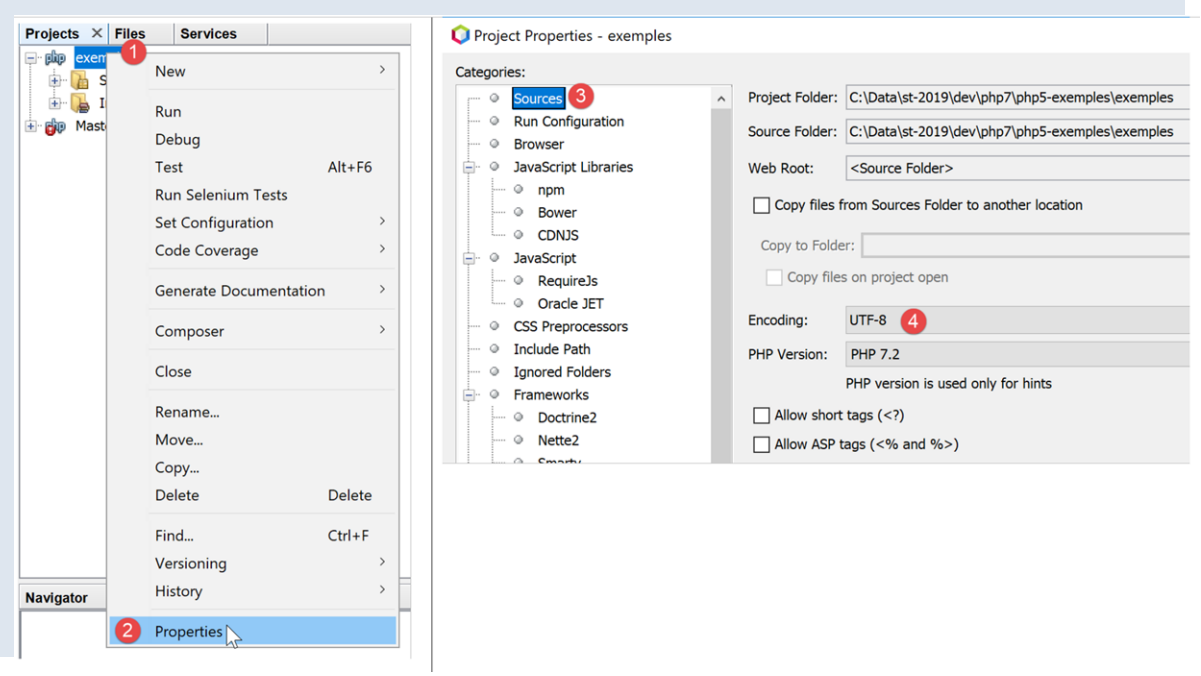
Nous allons écrire un script PHP qui exécute des ordres SQL trouvés dans le fichier texte **[creation.sql]** suivant :

```
1.  drop table if exists personnes
2.  SET NAMES 'utf8'
3.  create table personnes (prenom varchar(30) not null, nom varchar(30) not null, age integer not null, primar
    y key (nom,prenom))
4.  insert into personnes (prenom, nom, age) values('Paul','Langevin',48)
5.  insert into personnes (prenom, nom, age) values ('Sylvie','Lefur',70)
6.  insert into personnes (prenom, nom, age) values ('Sylvie','Lefur',70)
7.  insert into personnes (prenom, nom, age) values ('Pierre','Nicazou',35)
8.  insert into personnes (prenom, nom, age) values ('Géraldine','Colou',26)
9.  insert into personnes (prenom, nom, age) values ('Paulette','Girond',56)
10. insert into personnes (prenom, nom, age) values ('Paulette','Girond',56)
```

Commentaires

- le langage SQL (Structured Query Language) n'est pas sensible à la casse (majuscules, minuscules) des ordre SQL ;

- ligne 1 : on supprime la table **[personnes]** si elle existe ;
- ligne 2 : on indique au serveur MySQL qu'on va lui envoyer des caractères codés en UTF-8. Cet ordre SQL propre à MySQL est nécessaire ici par exemple pour avoir ligne 7, le é de Géraldine dans la base. Si on ne met pas la ligne 2, le é va être traduit en une suite de deux caractères étranges. Le client est le script PHP écrit sous Netbeans. Or celui-ci code les fichiers en UTF-8 **[1-4]** ci-dessous :



- ligne 3 : création de la table **[personnes]** avec les trois colonnes (prenom, nom, age) et la clé primaire (nom, prenom) ;
- lignes 4-10 : insertion de 7 lignes dans la table **[personnes]** ;
- ligne 6 : cet ordre d'insertion devrait échouer car il tente la même insertion que ligne 5. La contrainte de clé primaire devrait empêcher cette insertion : on ne peut avoir deux personnes ayant mêmes nom et prénom ;
- ligne 10 : cet ordre d'insertion devrait échouer car il tente la même insertion que ligne 9 ;

Le script PHP chargé d'exécuter les ordres SQL de ce fichier texte est le suivant **[mysql-03.php]** :

```
1.  <?php
2.
3.  // identité de la base de données
4.  const DSN = "mysql:host=localhost;dbname=dbpersonnes";
5.  // identifiants de l'utilisateur
6.  const ID = "admpersonnes";
7.  const PWD = "nobody";
8.  // identité du fichier texte des commandes SQL à exécuter
9.  const SQL_COMMANDS_FILENAME = "creation.sql";
10.
11. // ouverture connexion à la base MySql
12. try {
13.   $connexion = new PDO(DSN, ID, PWD);
14. } catch (PDOException $ex) {
15.   // affichage erreur
16.   print "Erreur : " . $ex->getMessage() . "\n";
17.   exit;
18. }
19. // on veut qu'à chaque erreur de SGBD, une exception soit lancée
20. $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
21. // exécution du fichier d'ordres SQL
22. $erreurs = exécuterCommandes($connexion, SQL_COMMANDS_FILENAME, TRUE, FALSE);
23. // fermeture connexion
24. $connexion = NULL;
25. //affichage nombre d'erreurs
26. printf("\n----------------------\nIl y a eu %d erreur(s)\n", count($erreurs));
27. for ($i = 0; $i < count($erreurs); $i++) {
28.   print "$erreurs[$i]\n";
29. }
30.
31. // c'est fini
32. print "Terminé\n";
33. exit;
34.
35. // ------------------------------------------------------------------------------
```

```
36.  function exécuterCommandes(PDO $connexion, string $SQLFileName, bool $suivi = FALSE, bool $arrêt = TRUE): array
37.  {
38.    // utilise la connexion $connexion
39.    // exécute les commandes SQL contenues dans le fichier texte SQLFileName
40.    // ce fichier est un fichier de commandes SQL à exécuter à raison d'une par ligne
41.    // si $suivi=1 alors chaque exécution d'un ordre SQL fait l'objet d'un affichage indiquant sa réussite ou son échec
42.    // si $arrêt=1, la fonction s'arrête sur la 1ère erreur rencontrée sinon elle exécute ttes les commandes sql
43.    // la fonction rend un tableau (nb d'erreurs, erreur1, erreur2, ...)
44.    // on vérifie la présence du fichier SQLFileName
45.
46.    if (!file_exists($SQLFileName)) {
47.      return ["Le fichier [$SQLFileName] n'existe pas"];
48.    }
49.
50.    // exécution des requêtes SQL contenues dans SQLFileName
51.    // on les met dans un tableau
52.    $requêtes = file($SQLFileName);
53.    // erreur ?
54.    if ($requêtes === FALSE) {
55.      return ["Erreur lors de l'exploitation du fichier SQL [$SQLFileName]"];
56.    }
57.    // on exécute les requêtes une par une - au départ pas d'erreurs
58.    $erreurs = [];
59.    $i = 0;
60.    $fini = FALSE;
61.    while ($i < count($requêtes) && !$fini) {
62.      // on récupère le texte de la requête
63.      // trim va enlever la marque de fin de ligne
64.      $requête = trim($requêtes[$i]);
65.      // requête vide ?
66.      if (strlen($requête) == 0) {
67.        // on ignore la requête et on passe à la requête suivante
68.        $i++;
69.        continue;
70.      }
71.      try {
72.        // exécution de la requête - une exception peut être lancée
73.        $connexion->exec($requête);
74.        // suivi écran ou non ?
75.        if ($suivi) {
76.          print "$requête : Exécution réussie\n";
77.        }
78.      } catch (PDOException $ex) {
79.        // il s'est produit une erreur
80.        addError($erreurs, $requête, $ex->getMessage(), $suivi);
81.        // est-ce qu'on s'arrête ?
82.        $fini = $arrêt;
83.      }
84.      // requête suivante
85.      $i++;
86.    }
87.    // résultat
88.    return $erreurs;
89.  }
90.
91.  function addError(array &$erreurs, string $requête, string $msg, bool $suivi): void
92.  {
93.    // on ajoute un msg d'erreur
94.    $msg = "$requête : Erreur (" . $msg . ")";
95.    $erreurs[] = $msg;
96.    // suivi écran ou non ?
97.    if ($suivi) {
98.      print "$msg\n";
99.    }
100. }
```

Commentaires

- la fonction **[exécuterCommandes]** (lignes 36-89) est chargée d'exécuter les commandes SQL qu'elle trouve dans le fichier texte **[$SQLFileName]** (paramètre 2). Pour les exécuter elle utilise la connexion ouverte **[$connexion]** (paramètre 1) avec le serveur MySQL. Le troisième paramètre **[$suivi]** est un booléen qui contrôle les affichages écran : à TRUE, l'ordre SQL exécuté est affiché à l'écran avec sa réussite ou son échec, sinon l'exécution de l'ordre SQL est silencieux. Le quatrième paramètre **[$arrêt]** contrôle ce qu'il faut faire lorsqu'une commande SQL échoue : à TRUE, il indique que l'exécution des commandes SQL doit s'arrêter, sinon celle-ci continue. La fonction **[exécuterCommandes]** rend un tableau de messages d'erreurs, vide s'il n'y a pas eu d'erreurs ;
- lignes 11-18 : on ouvre la connexion vers la base MySQL **[dbpersonnes]**. Si l'ouverture échoue, un message d'erreur est affiché et on s'arrête (lignes 14-18) ;
- ligne 22 : on passe donc une connexion ouverte à la fonction **[exécuterCommandes]**. Elle sera fermée au retour de la fonction (ligne 24) ;
- ligne 20 : avant de la passer à la fonction **[exécuterCommandes]**, on configure la connexion. En cas d'erreur, les opérations SQL avec un objet **[PDO]** peuvent soit rendre le booléen **FALSE** (valeur par défaut), soit lancer une exception. La ligne 20 choisit

ce second cas. En effet, il est facile 'd'oublier' de vérifier le résultat booléen de l'exécution d'un ordre SQL. Cela produira une erreur ultérieurement mais ailleurs dans le code rendant ainsi plus difficile le lieu originel de celle-ci. Dans le cas d'une exception non gérée (absence de *catch*), l'exception va remonter dans le code jusqu'à rencontrer un *catch* ou jusqu'à remonter à l'interpréteur PHP qui lui interceptera l'exception. Dans ce cas, la nature de l'exception et son lieu d'origine dans le code sont affichés ;

- ligne 22 : la fonction **[exécuterCommandes]** est appelée pour exécuter le fichier d'ordres SQL **[$SQLFileName]** ;
- lignes 46-48 : on vérifie que le fichier des ordres SQL existe bien. Si ce n'est pas le cas, on note l'erreur et on retourne ce résultat ;
- ligne 52 : on met les ordres SQL dans un tableau **[$requêtes]**. Lignes 53-55, si l'opération échoue, on rend un tableau d'erreurs avec un unique message ;
- ligne 58 : on va cumuler les erreurs dans le tableau **[$erreurs]** ;
- ligne 59 : n° de la requête ;
- ligne 60 : le booléen **[$fini]** contrôle l'exécution des ordres SQL du tableau **[$requêtes]**. Lorsqu'il passe à TRUE, l'exécution s'arrête ;
- ligne 61 : on boucle sur toutes les requêtes ;
- ligne 64 : on extrait le texte de l'ordre SQL n° i. La fonction **[trim]** va enlever les espaces qui précèdent et suivent le texte de l'ordre SQL. Par 'espaces', il faut entendre le blanc \b, le retour chariot \r, la marque de fin de ligne \n, le saut de page \f, la tabulation \t… Ce qui nous importe ici c'est que la marque de fin de ligne du texte SQL va disparaître ;
- lignes 66-70 : si le texte SQL est vide, alors on ignore la requête et on passe à la suivante ;
- ligne 73 : on envoie l'ordre SQL au serveur MySQL. La méthode **[PDO::exec]** va lancer une exception si l'exécution échoue. On rappelle que ce comportement est dû à la configuration faite à la ligne 20 ;
- ligne 80 : le message d'erreur est ajouté au tableau des erreurs ;
- ligne 82 : on positionne le booléen **[$fini]** qui contrôle la boucle. Si le paramètre **[$arrêt]** (ligne 36) vaut TRUE, on doit arrêter la boucle ;
- lignes 75-77 : si l'exécution de l'ordre SQL a réussi, on l'affiche à l'écran si le paramètre **[$suivi]** (ligne 36) vaut TRUE ;
- ligne 88 : une fois toutes les ordres SQL exécutés, on rend le tableau des erreurs **[$erreurs]** ;

La fonction **[adError]** des lignes 91-99 permet d'ajouter une erreur au tableau des erreurs **[$erreurs]** :

- ligne 91 : la fonction reçoit 4 paramètres :
  - le paramètre **[$erreurs]** est passé par référence. En effet on veut agir sur le tableau qui est passé en paramètre et non sur une copie de celui-ci ;
  - le paramètre **[$requête]** est le texte SQL de l'ordre qui a échoué ;
  - le paramètre **[$msg]** est le message d'erreur lié à l'ordre qui a échoué ;
  - le booléen **[$suivi]** indique si le message d'erreur doit être affiché ($suivi=TRUE) ou non ($suivi=FALSE) sur la console ;

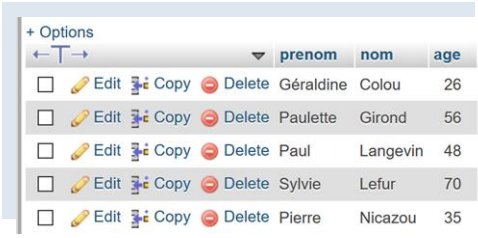
La fonction **[exécuterCommandes]** est appelé par le script des lignes 3-33 :

- lignes 11-18 : une connexion est faite avec la base de données MySQL **[dbpersonnes]** ;
- ligne 20 : la connexion est configurée ;
- ligne 22 : le fichier des ordres SQL est ensuite exécuté ;
- ligne 24 : on ferme la connexion ;
- lignes 26-29 : on affiche les erreurs rendues par la fonction **[exécuterCommandes]** ;

Les **résultats** écran :

```
1   drop table if exists personnes : Exécution réussie
2   SET NAMES 'utf8' : Exécution réussie
3   create table personnes (prenom varchar(30) not null, nom varchar(30) not null, age integer not null,
    primary key (nom,prenom)) : Exécution réussie
4   insert into personnes (prenom, nom, age) values('Paul','Langevin',48) : Exécution réussie
5   insert into personnes (prenom, nom, age) values ('Sylvie','Lefur',70) : Exécution réussie
6   insert into personnes (prenom, nom, age) values ('Sylvie','Lefur',70) : Erreur (SQLSTATE[23000]: Integrity
    constraint violation: 1062 Duplicate entry 'Lefur-Sylvie' for key 'PRIMARY')
7   insert into personnes (prenom, nom, age) values ('Pierre','Nicazou',35) : Exécution réussie
8   insert into personnes (prenom, nom, age) values ('Géraldine','Colou',26) : Exécution réussie
9   insert into personnes (prenom, nom, age) values ('Paulette','Girond',56) : Exécution réussie
10  insert into personnes (prenom, nom, age) values ('Paulette','Girond',56) : Erreur (SQLSTATE[23000]:
    Integrity constraint violation: 1062 Duplicate entry 'Girond-Paulette' for key 'PRIMARY')
11
12  ----------------------
13  Il y a eu 2 erreur(s)
14  insert into personnes (prenom, nom, age) values ('Sylvie','Lefur',70) : Erreur (SQLSTATE[23000]: Integrity
    constraint violation: 1062 Duplicate entry 'Lefur-Sylvie' for key 'PRIMARY')
15  insert into personnes (prenom, nom, age) values ('Paulette','Girond',56) : Erreur (SQLSTATE[23000]:
    Integrity constraint violation: 1062 Duplicate entry 'Girond-Paulette' for key 'PRIMARY')
16  Terminé
```

Les insertions faites sont visibles avec phpMyAdmin :



1.12.5 Exécution d'ordres SQL quelconques

Le script suivant montre l'exécution des ordres SQL du fichier texte **[sql.txt]** suivant :

```
set names 'utf8'
select * from personnes
select nom,prenom from personnes order by nom asc, prenom desc
select * from personnes where age between 20 and 40 order by age desc, nom asc, prenom asc
insert into personnes values('Josette','Bruneau',46)
update personnes set age=47 where nom='Bruneau'
select * from personnes where nom='Bruneau'
delete from personnes where nom='Bruneau'
select * from personnes where nom='Bruneau'
xselect * from personnes where nom='Bruneau'
```

Parmi ces ordres SQL, il y a l'ordre **select** qui ramène des résultats de la base de données, les ordres **insert**, **update**, **delete** qui modifient la base sans ramener de résultats et enfin des ordres erronés tels que le dernier (**xselect**). Le script **[mysql-04.php]** est le suivant :

```
<?php

// identité de la base de données
const DSN = "mysql:host=localhost;dbname=dbpersonnes";
// identifiants de l'utilisateur
const ID = "admpersonnes";
const PWD = "nobody";
// identité du fichier texte des commandes SQL à exécuter
const SQL_COMMANDS_FILENAME = "sql.txt";

try {
  // connexion à la base MySql
  $connexion = new PDO(DSN, ID, PWD);
} catch (PDOException $ex) {
  // affichage erreur
  print "Erreur : " . $ex->getMessage() . "\n";
  exit;
}
// on veut qu'à chaque erreur de SGBD, une exception soit lancée
$connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// exécution du fichier d'ordres SQL
$erreurs = exécuterCommandes($connexion, SQL_COMMANDS_FILENAME, TRUE, FALSE);
// fermeture connexion
$connexion = NULL;
//affichage nombre d'erreurs
printf("\n----------------------\nIl y a eu %d erreur(s)\n", count($erreurs));
for ($i = 0; $i < count($erreurs); $i++) {
  print "$erreurs[$i]\n";
}

// c'est fini
print "Terminé\n";
exit;

// ------------------------------------------------------------------------------
function exécuterCommandes(PDO $connexion, string $SQLFileName, bool $suivi = FALSE, bool $arrêt = TRUE): array {
…
  // on exécute les requêtes une par une - au départ pas d'erreurs
  $erreurs = [];
  $i = 0;
  $fini = FALSE;
  while ($i < count($requêtes) && !$fini) {
    // on récupère le texte de la requête
    // trim va enlever la marque de fin de ligne
    $requête = trim($requêtes[$i]);
    // requête vide ?
    if (strlen($requête) == 0) {
      // on ignore la requête et on passe à la requête suivante
      $i++;
```

```
50.         continue;
51.       }
52.       // exécution de la requête
53.       // on récupère son nom
54.       $commande = "";
55.       if (preg_match("/^\s*(\S+)/", $requête, $champs)) {
56.         $commande = strtolower($champs[0]);
57.       }
58.       try {
59.         // est-ce un ordre SELECT ?
60.         if ($commande === "select") {
61.           $résultat = $connexion->query($requête);
62.         } else {
63.           $résultat = $connexion->exec($requête);
64.         }
65.         // suivi écran ou non ?
66.         if ($suivi) {
67.           print "[$requête] : Exécution réussie\n";
68.         }
69.         // on affiche le résultat de l'exécution
70.         afficherInfos($commande, $résultat);
71.       } catch (PDOException $ex) {
72.         // il s'est produit une erreur
73.         addError($erreurs, $requête, $ex->getMessage(), $suivi);
74.         // est-ce qu'on s'arrête ?
75.         $fini = $arrêt;
76.       }
77.       // requête suivante
78.       $i++;
79.     }
80.     // résultat
81.     return $erreurs;
82.   }
83.
84.   function addError(array &$erreurs, string $requête, string $msg, bool $suivi): void {
85.     …
86.   }
87.
88.   // ------------------------------------------------------------------------------
89.   function afficherInfos(string $commande, $résultat): void {
90.     // affiche le résultat $résultat d'une requête sql
91.     // s'agissait-il d'un select ?
92.     switch ($commande) {
93.       case "select" :
94.         // on affiche les noms des champs
95.         $titre = "";
96.         $nbColonnes = $résultat->columnCount();
97.         for ($i = 0; $i < $nbColonnes; $i++) {
98.           $infos = $résultat->getColumnMeta($i);
99.           $titre .= $infos['name'] . ",";
100.        }
101.        // on enlève le dernier caractère ,
102.        $titre = substr($titre, 0, strlen($titre) - 1);
103.        // on affiche la liste des champs
104.        print "$titre\n";
105.        // ligne séparatrice
106.        $séparateurs = "";
107.        for ($i = 0; $i < strlen($titre); $i++) {
108.          $séparateurs .= "-";
109.        }
110.        print "$séparateurs\n";
111.        // données
112.        foreach ($résultat as $ligne) {
113.          $data = "";
114.          for ($i = 0; $i < $nbColonnes; $i++) {
115.            $data .= $ligne[$i] . ",";
116.          }
117.          // on enlève le dernier caractère ,
118.          $data = substr($data, 0, strlen($data) - 1);
119.          // on affiche
120.          print "$data\n";
121.        }
122.        break;
123.      case "update":
124.      case "insert":
125.      case "delete";
126.        print " $résultat lignes(s) a (ont) été modifiée(s)\n";
127.        break;
128.    }
129. }
```

Commentaires

- lignes 36-83 : la fonction **[exécuterCommandes]** est légèrement modifiée : l'ordre SQL **[select]** ne s'exécute pas de la même façon que les autres ordres SQL. Cet ordre est le seul à ramener comme résultat une table, ç-à-d un ensemble de lignes et de colonnes de la base de données ;

- lignes 55-57 : on isole le 1er mot de l'ordre SQL à l'aide d'une expression régulière ;
- lignes 60-64 : si la commande SQL est **[select]**, on utilise la méthode **[PDO::query]** sinon la méthode **[PDO::exec]** pour exécuter l'ordre SQL. Dans les deux cas, si l'exécution échoue, une exception sera lancée et interceptée lignes 71-77. Si l'exécution réussit, la ligne 70 affiche son résultat ;
- lignes 90-130 : la fonction *afficherInfos* affiche des informations sur le résultat de l'exécution d'un ordre SQL ;
- ligne 94 : on traite le cas du **[select]**. Son résultat est un objet de type **[PDOStatement]** ;
- ligne 96 : la méthode **[PDOStatement::getColumnCount()]** rend le nombre de colonnes de la table résultat du *select* ;
- lignes 98-99 : la méthode **[PDOStatement::getMeta(i)]** rend un dictionnaire d'informations sur la colonne n° *i* de la table résultat du *select*. Dans ce dictionnaire, la valeur associée à la clé '*name*' est le nom de la colonne ;
- lignes 97-102 : les noms des colonnes de la table résultat du *select* sont concaténées dans une chaîne de caractères ;
- lignes 105-110 : on construit une ligne de séparation ayant la même longueur que la chaîne de caractères construite précédemment ;
- lignes 112-121 : un objet de type *PDOStatement* peut être parcouru par une boucle *foreach*. A chaque itération, l'élément obtenu est une ligne de la table résultat du *select* sous la forme d'un tableau de valeurs représentant les valeurs des différentes colonnes de la ligne. On affiche toutes ces valeurs avec une boucle *for* (lignes 114-116) ;
- lignes 123-127 : le résultat de l'exécution d'un ordre *insert*, *update*, *delete* est le nombre de lignes modifiées par l'ordre ;

Les **résultats** écran :

```
1   [set names 'utf8'] : Exécution réussie
2   [select * from personnes] : Exécution réussie
3   prenom,nom,age
4   --------------
5   Géraldine,Colou,26
6   Paulette,Girond,56
7   Paul,Langevin,48
8   Sylvie,Lefur,70
9   Pierre,Nicazou,35
10  [select nom,prenom from personnes order by nom asc, prenom desc] : Exécution réussie
11  nom,prenom
12  ----------
13  Colou,Géraldine
14  Girond,Paulette
15  Langevin,Paul
16  Lefur,Sylvie
17  Nicazou,Pierre
18  [select * from personnes where age between 20 and 40 order by age desc, nom asc, prenom asc] : Exécution
    réussie
19  prenom,nom,age
20  --------------
21  Pierre,Nicazou,35
22  Géraldine,Colou,26
23  [insert into personnes values('Josette','Bruneau',46)] : Exécution réussie
24   1 lignes(s) a (ont) été modifiée(s)
25  [update personnes set age=47 where nom='Bruneau'] : Exécution réussie
26   1 lignes(s) a (ont) été modifiée(s)
27  [select * from personnes where nom='Bruneau'] : Exécution réussie
28  prenom,nom,age
29  --------------
30  Josette,Bruneau,47
31  [delete from personnes where nom='Bruneau'] : Exécution réussie
32   1 lignes(s) a (ont) été modifiée(s)
33  [select * from personnes where nom='Bruneau'] : Exécution réussie
34  prenom,nom,age
35  --------------
36  [insert into personnes values('Josette','Bruneau',46)] : Exécution réussie
37   1 lignes(s) a (ont) été modifiée(s)
38  [xselect * from personnes where nom='Bruneau'] : Erreur (SQLSTATE[42000]: Syntax error or access violation:
    1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
    for the right syntax to use near 'xselect * from personnes where nom='Bruneau'' at line 1)
39
40  ------------------------
41  Il y a eu 1 erreur(s)
42  [xselect * from personnes where nom='Bruneau'] : Erreur (SQLSTATE[42000]: Syntax error or access violation:
    1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
    for the right syntax to use near 'xselect * from personnes where nom='Bruneau'' at line 1)
43  Terminé
```

1.12.6 Utilisation d'ordres SQL préparés

1.12.6.1 Exemple 1

Examinons le script suivant **[mysql-05.php]** :

```
1.  <?php
2.
3.  // identité de la base de données
4.  const DSN = "mysql:host=localhost;dbname=dbpersonnes";
5.  // identifiants de l'utilisateur
6.  const ID = "admpersonnes";
7.  const PWD = "nobody";
8.
9.  try {
10.   // connexion à la base MySql
11.   $connexion = new PDO(DSN, ID, PWD);
12.   // on veut qu'à chaque erreur de SGBD, une exception soit lancée
13.   $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
14.   // on vide la table des personnes
15.   $connexion->exec("delete from personnes");
16.   // une liste de personnes
17.   $personnes = [];
18.   $personnes[] = ["nom" => "Langevin", "prenom" => "Paul", "age" => 47];
19.   $personnes[] = ["nom" => "Lefur", "prenom" => "Sylvie", "age" => 28];
20.   // on va mettre ces personnes dans la base de données
21.   $statement = $connexion-
    >prepare("insert into personnes (nom, prenom, age) values (:nom, :prenom, :age)");
22.   for ($i = 0; $i < count($personnes); $i++) {
23.     $statement->execute($personnes[$i]);
24.   }
25. } catch (PDOException $ex) {
26.   // affichage erreur
27.   print "Erreur : " . $ex->getMessage() . "\n";
28. } finally {
29. // fermeture connexion
30.   $connexion = NULL;
31. }
32.
33. // c'est fini
34. print "Terminé\n";
35. exit;
```

Commentaires

Nous nous intéressons ici aux lignes 16-24 qui insèrent deux personnes dans la table des personnes de la base **[dbpersonnes]**.

- ligne 21 : on 'prépare' un ordre SQL paramétré. Les paramètres sont précédés du caractère : :**nom**, :**prenom**, :**age**. Pour 'préparer' un ordre SQL, on utilise la méthode **[PDO::prepare]**. Le résultat est un type **[PDOStatement]**. La 'préparation' n'est pas une exécution : rien n'est exécuté ;
- ligne 23 : exécution de l'ordre 'préparé' avec la méthode **[PDOStatement::execute]**. Pour ce faire, il faut donner des valeurs aux paramètres :**nom**, :**prenom** et :**age**. Il y a plusieurs façons de faire cela. On utilise ici un dictionnaire ayant pour clés les paramètres de l'ordre préparé que l'on passe à la méthode **[PDOStatement::execute]**. Une autre façon de faire est de donner aux paramètres une valeur avec la méthode **[PDOStatement::bindValue($paramètre,$valeur)]**. Par exemple ici :

```
$statement→bindValue("nom","Langevin");
$statement→bindValue("prenom","Paul");
$statement→bindValue("age",47);
$statement→execute();
```

  L'inconvénient est qu'il faut réitérer cette instruction pour chaque paramètre. La méthode du dictionnaire peut alors être plus pratique. La méthode **[PDOStatement::execute]** rend FALSE si l'exécution échoue ;
- la méthode utilisée ici pour faire les insertions :
  - **une** méthode préparée ;
  - **n** exécutions de l'ordre préparé ;

  est plus économique en temps d'exécution que d'exécuter **n** ordres SQL **différents**. Cette méthode est donc à privilégier. Elle est utilisable pour les ordres SQL *select, update, delete, insert*. Dans le cas de l'ordre SQL *select*, après son exécution avec **[PDOStatement::execute]**, on récupère les lignes du résultat avec la méthode **[PDOStatement::fetchAll]** ;

1.12.6.2 Exemple 2

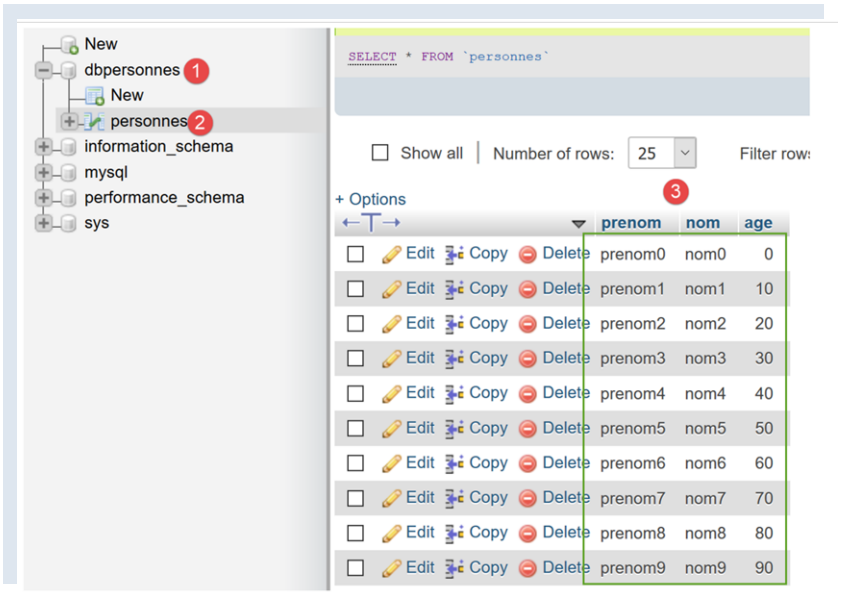
Le script suivant **[mysql-06.php]** montre l'utilisation d'un ordre préparé pour l'opération SQL **select**, ainsi que diverses façons de récupérer les lignes ramenées par cette opération :

```
1.  <?php
2.
3.  // identité de la base de données
```

```
4.  const DSN = "mysql:host=localhost;dbname=dbpersonnes";
5.  // identifiants de l'utilisateur
6.  const ID = "admpersonnes";
7.  const PWD = "nobody";
8.
9.  try {
10.   // connexion à la base MySql
11.   $connexion = new PDO(DSN, ID, PWD);
12.   // on veut qu'à chaque erreur de SGBD, une exception soit lancée
13.   $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
14.   // on vide la table des personnes
15.   $connexion->exec("delete from personnes");
16.   // on va mettre ces personnes dans la base de données
17.   $statement = $connexion-
    >prepare("insert into personnes (nom, prenom, age) values (:nom, :prenom, :age)");
18.   for ($i = 0; $i < 10; $i++) {
19.     $statement->execute(["nom" => "nom" . $i, "prenom" => "prenom" . $i, "age" => $i * 10]);
20.   }
21.   // on interroge la base
22.   $statement = $connexion->prepare("select nom, prenom, age from personnes");
23.   $statement->execute();
24.   // 1ère ligne
25.   $ligne = $statement->fetch();
26.   var_dump($ligne);
27.   // 2ième ligne
28.   $ligne = $statement->fetch(PDO::FETCH_ASSOC);
29.   var_dump($ligne);
30.   // 3ième ligne
31.   $ligne = $statement->fetch(PDO::FETCH_OBJ);
32.   var_dump($ligne);
33.   // 4ième ligne
34.   $statement->setFetchMode(PDO::FETCH_CLASS, "Person");
35.   $ligne = $statement->fetch();
36.   var_dump($ligne);
37.   // lecture séquentielle de toutes les lignes
38.   $statement = $connexion->prepare("select nom, prenom, age from personnes");
39.   $statement->execute();
40.   $statement->setFetchMode(PDO::FETCH_CLASS, "Person");
41.   while ($personne = $statement->fetch()) {
42.     print "$personne\n";
43.   }
44. } catch (PDOException $ex) {
45.   // affichage erreur
46.   print "Erreur : " . $ex->getMessage() . "\n";
47. } finally {
48. // fermeture connexion
49.   $connexion = NULL;
50. }
51.
52. // c'est fini
53. print "Terminé\n";
54. exit;
55.
56. class Person {
57.   private $nom;
58.   private $prenom;
59.   private $age;
60.
61.   public function __toString() {
62.     return "Personne[$this->nom,$this->prenom,$this->age]";
63.   }
64.
65. }
```

Commentaires

- lignes 17-20 : on insère 10 lignes dans la table **[personnes]** de la base **[admpersonnes]** :



- ligne 22 : on « prépare » un ordre SQL **[select]** qu'on exécute ligne 23 ;
- ligne 25 : on récupère avec la méthode **[PDOStatement::fetch]** une ligne du résultat de l'opération SQL **[select]** exécutée. La méthode **[PDOStatement::fetch]** peut récupérer de diverses façons les lignes résultats d'une opération SQL **[select]** préparée. Le script en présente quelques unes. La méthode **[PDOStatement::fetch]** sans paramètres ramène la ligne courante du **[select]** sous la forme d'un dictionnaire indexé à la fois sur les n°s de colonnes et leurs noms ;
- ligne 26 : affiche le résultat suivant :

```
1   array(6) {
2     ["nom"]=>
3     string(4) "nom0"
4     [0]=>
5     string(4) "nom0"
6     ["prenom"]=>
7     string(7) "prenom0"
8     [1]=>
9     string(7) "prenom0"
10    ["age"]=>
11    string(1) "0"
12    [2]=>
13    string(1) "0"
14  }
```

- lignes 28-29 : le paramètre **[PDO::FETCH_ASSOC]** fait que la ligne ramenée est un dictionnaire indexé par les noms des colonnes de la table :

```
1   array(3) {
2     ["nom"]=>
3     string(4) "nom1"
4     ["prenom"]=>
5     string(7) "prenom1"
6     ["age"]=>
7     string(2) "10"
8   }
```

- lignes 31-32 : le paramètre **[PDO::FETCH_OBJ]** fait que la ligne ramenée est un objet de type **[stdclass]** dont les attributs sont les noms des colonnes de la table :

```
1   object(stdClass)#2 (3) {
2     ["nom"]=>
3     string(4) "nom2"
4     ["prenom"]=>
5     string(7) "prenom2"
6     ["age"]=>
7     string(2) "20"
8   }
```

- ligne 34 : on fixe le mode de recherche de la méthode **[fetch]** avec la méthode **[PDOStatement::setFetchMode]**. Ce mode devient alors le mode par défaut tant qu'il n'est pas changé soit par une autre opération **[PDOStatement::setFetchMode]** soit en passant un mode en paramètre à la méthode **[PDOStatement::fetch]** comme il a été fait précédemment. L'opération **[setFetchMode(PDO::FETCH_CLASS, "Person")]** indique que la ligne lue doit être placée dans un objet de type **[Person]**. Cette classe doit avoir parmi ces attributs, des attributs portant le nom des colonnes de la ligne lue. C'est le cas de la classe **[Person]** définie aux lignes 56-63 ;
- la ligne 36 affiche le résultat suivant :

```
1   object(Person)#4 (3) {
2     ["nom":"Person":private]=>
3     string(4) "nom3"
4     ["prenom":"Person":private]=>
5     string(7) "prenom3"
6     ["age":"Person":private]=>
7     string(2) "30"
8   }
```

- lignes 38-43 : montrent comment exploiter séquentiellement les résultats du **[select]** ;
- ligne 42 : l'affichage de **[$personne]** va utiliser la méthode **[__toString]** de la classe **[Person]** ;

1.12.7 Utilisation de transactions

Une transaction permet de regrouper une séquence d'ordres SQL en une unité d'exécution : soit tous les ordres réussissent soit l'un d'eux échoue et alors tous les ordres SQL qui ont précédé celui-ci sont annulés. Dit autrement, lorsqu'on utilise une transaction pour exécuter des ordres SQL, après l'exécution de celle-ci la base de données est dans un état **stable** :

- soit dans un **état nouveau** créé par l'exécution réussie de tous les ordres SQL de la transaction ;
- soit dans **l'état dans laquelle elle était** avant que la transaction ne commence à être exécutée ;

Nous allons reprendre l'exemple de l'exécution des ordres SQL contenus dans un fichier texte étudié au paragraphe lien. Nous allons inclure cette exécution dans une transaction. Les ordres SQL seront contenus dans le fichier **[sql2.txt]** suivant :

```
1.  set names 'utf8'
2.  select * from personnes
3.  select nom,prenom from personnes order by nom asc, prenom desc
4.  select * from personnes where age between 20 and 40 order by age desc, nom asc, prenom asc
5.  insert into personnes values('Josette','Bruneau',46)
6.  update personnes set age=47 where nom='Bruneau'
7.  select * from personnes where nom='Bruneau'
8.  delete from personnes where nom='Bruneau'
9.  select * from personnes where nom='Bruneau'
10. insert into personnes values('Josette','Bruneau',46)
11. select * from personnes where nom='Bruneau'
12. xselect * from personnes where nom='Bruneau'
```

L'ordre erroné de la ligne 12 va faire échouer toute la transaction. On devrait donc retrouver la base comme elle était avant la transaction. Dans l'exemple ci-dessus, on ne devrait donc pas voir dans la table la ligne insérée par la ligne 10 ci-dessus. Le script évolue très peu. On redonne cependant la totalité du code **[mysql-07.php]** :

```
1.  <?php
2.
3.  // identité de la base de données
4.  const DSN = "mysql:host=localhost;dbname=dbpersonnes";
5.  // identifiants de l'utilisateur
6.  const ID = "admpersonnes";
7.  const PWD = "nobody";
8.  // identité du fichier texte des commandes SQL à exécuter
9.  const SQL_COMMANDS_FILENAME = "sql2.txt";
10.
11. try {
12.   // connexion à la base MySql
13.   $connexion = new PDO(DSN, ID, PWD);
14. } catch (PDOException $ex) {
15.   // affichage erreur
16.   print "Erreur : " . $ex->getMessage() . "\n";
17.   exit;
18. }
19. // on veut qu'à chaque erreur de SGBD, une exception soit lancée
20. $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
21. // exécution du fichier d'ordres SQL
22. $erreurs = exécuterCommandes($connexion, SQL_COMMANDS_FILENAME, TRUE);
23. // fermeture connexion
24. $connexion = NULL;
25. //affichage nombre d'erreurs
26. printf("\n----------------------\nIl y a eu %d erreur(s)\n", count($erreurs));
```

```
for ($i = 0; $i < count($erreurs); $i++) {
  print "$erreurs[$i]\n";
}

// c'est fini
print "Terminé\n";
exit;

// ------------------------------------------------------------------------------
function exécuterCommandes(PDO $connexion, string $SQLFileName, bool $suivi = FALSE): array {
// utilise la connexion $connexion
// exécute les commandes SQL contenues dans le fichier texte SQLFileName
// ce fichier est un fichier de commandes SQL à exécuter à raison d'une par ligne
// les commandes SQL sont exécutées dans une transaction
// si l'un des ordres échoue, la transaction est annulée et on retrouve la base de données dans l'état où elle était avant la transaction
// si $suivi=1 alors chaque exécution d'un ordre SQL fait l'objet d'un affichage indiquant sa réussite ou son échec
// la fonction rend un tableau (nb d'erreurs, erreur1, erreur2, ...)
//
// on vérifie la présence du fichier SQLFileName
  if (!file_exists($SQLFileName)) {
    return ["Le fichier [$SQLFileName] n'existe pas"];
  }
  // exécution des requêtes SQL contenues dans SQLFileName
  // on les met dans un tableau
  $requêtes = file($SQLFileName);
  // erreur ?
  if ($requêtes === FALSE) {
    return ["Erreur lors de l'exploitation du fichier SQL [$SQLFileName]"];
  }
  // les requêtes vont être placées dans une transaction
  $connexion->beginTransaction();
  // on exécute les requêtes une par une - au départ pas d'erreurs
  $erreurs = [];
  $i = 0;
  $fini = FALSE;
  while ($i < count($requêtes) && !$fini) {
    // on récupère le texte de la requête
    // trim va enlever la marque de fin de ligne
    $requête = trim($requêtes[$i]);
    // requête vide ?
    if (strlen($requête) == 0) {
      // on ignore la requête et on passe à la requête suivante
      $i++;
      continue;
    }
    // exécution de la requête
    // on récupère son nom
    $commande = "";
    if (preg_match("/^\s*(\S+)/", $requête, $champs)) {
      $commande = strtolower($champs[0]);
    }
    try {
      // est-ce un ordre SELECT ?
      if ($commande === "select") {
        $résultat = $connexion->query($requête);
      } else {
        $résultat = $connexion->exec($requête);
      }
      // suivi écran ou non ?
      if ($suivi) {
        print "[$requête] : Exécution réussie\n";
      }
      // on affiche le résultat de l'exécution
      afficherInfos($commande, $résultat);
    } catch (PDOException $ex) {
      // il s'est produit une erreur
      addError($erreurs, $requête, $ex->getMessage(), $suivi);
      // on s'arrête au tour suivant
      $fini = TRUE;
    }
    // requête suivante
    $i++;
  }
  // fin de la transaction
  if (!$fini) {
    // il n'y a pas eu d'erreurs : on valide la transaction
    $connexion->commit();
  } else {
    // il y a eu des erreurs : on annule la transaction
    $connexion->rollBack();
    // ajout erreur
    addError($erreurs, "", "Transaction annulée", $suivi);
  }
  // résultat
  return $erreurs;
}
```

```
113.
114. function addError(array &$erreurs, string $requête, string $msg, bool $suivi): void {
115.   …
116. }
117.
118. // ----------------------------------------------------------------------------
119. function afficherInfos(string $commande, $résultat): void {
120.   …
121.   }
122. }
```

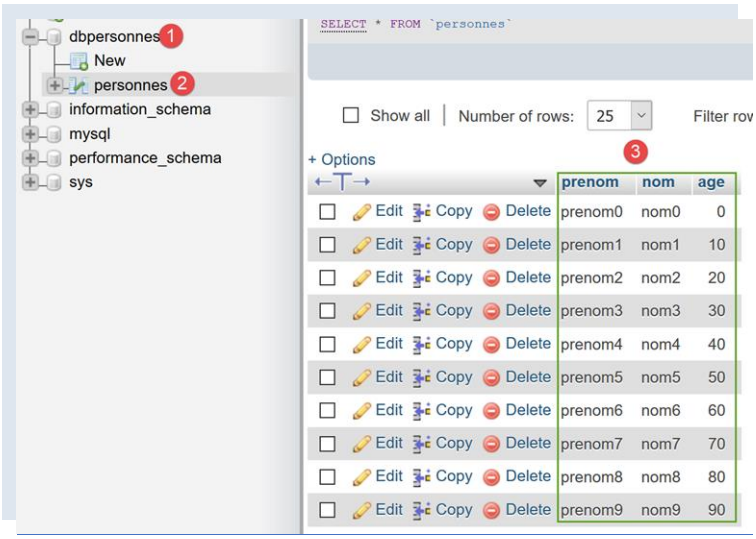
Commentaires

Nous avons surligné les modifications du script original **[mysql-04.php]**.

- lignes 22, 36 : la fonction **[exécuterCommandes]** a perdu son quatrième paramètre **[$arrêt=TRUE]**. En effet, comme les ordres SQL s'exécutent au sein d'une transaction, toute erreur provoquera l'arrêt de la transaction ;
- lignes 40-41 : rappel de la fonction d'une transaction ;
- ligne 57 : on démarre une transaction. A partir de maintenant, tout ordre SQL exécuté dans la boucle des lignes 62-99 l'est au sein de cette transaction ;
- lignes 101-109 : le booléen **[$fini]** est à TRUE s'il y a eu erreur (ligne 95). Lorsqu'il est à FALSE, il n'y a pas eu d'erreurs et on valide alors la transaction (ligne 103). Lorsqu'il est à TRUE, il y a eu des erreurs et on annule alors la transaction (ligne 106) et on rajoute l'erreur de transaction dans la liste des erreurs (ligne 108) ;

Résultats

Avant exécution du script, la base **[admpersonnes]** est dans l'état suivant :



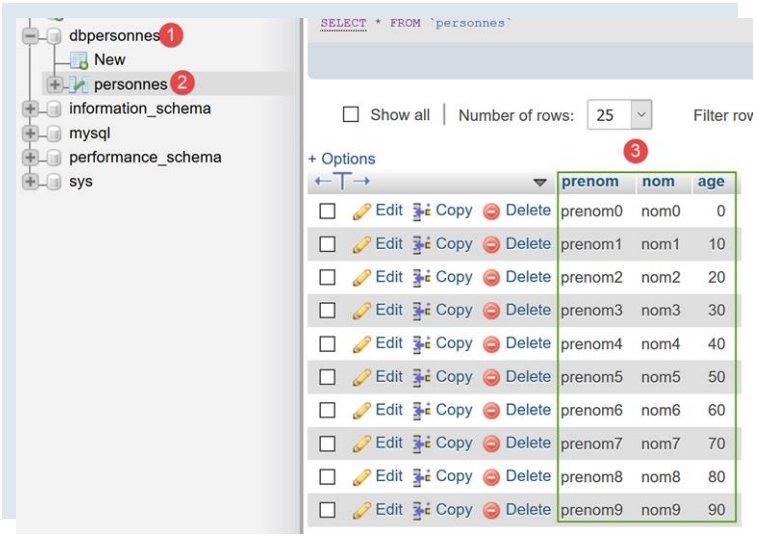
On exécute le script **[mysql-07.php]**. Les affichages écran sont alors les suivants :

```
1   [set names 'utf8'] : Exécution réussie
2   [select * from personnes] : Exécution réussie
3   prenom,nom,age
4   --------------
5   prenom0,nom0,0
6   prenom1,nom1,10
7   prenom2,nom2,20
8   prenom3,nom3,30
9   prenom4,nom4,40
10  prenom5,nom5,50
11  prenom6,nom6,60
12  prenom7,nom7,70
13  prenom8,nom8,80
14  prenom9,nom9,90
15  [select nom,prenom from personnes order by nom asc, prenom desc] : Exécution réussie
16  nom,prenom
17  ----------
18  nom0,prenom0
19  nom1,prenom1
20  nom2,prenom2
21  nom3,prenom3
```

```
22  nom4,prenom4
23  nom5,prenom5
24  nom6,prenom6
25  nom7,prenom7
26  nom8,prenom8
27  nom9,prenom9
28  [select * from personnes where age between 20 and 40 order by age desc, nom asc, prenom asc] : Exécution
    réussie
29  prenom,nom,age
30  --------------
31  prenom4,nom4,40
32  prenom3,nom3,30
33  prenom2,nom2,20
34  [insert into personnes values('Josette','Bruneau',46)] : Exécution réussie
35   1 lignes(s) a (ont) été modifiée(s)
36  [update personnes set age=47 where nom='Bruneau'] : Exécution réussie
37   1 lignes(s) a (ont) été modifiée(s)
38  [select * from personnes where nom='Bruneau'] : Exécution réussie
39  prenom,nom,age
40  --------------
41  Josette,Bruneau,47
42  [delete from personnes where nom='Bruneau'] : Exécution réussie
43   1 lignes(s) a (ont) été modifiée(s)
44  [select * from personnes where nom='Bruneau'] : Exécution réussie
45  prenom,nom,age
46  --------------
47  [insert into personnes values('Josette','Bruneau',46)] : Exécution réussie
48   1 lignes(s) a (ont) été modifiée(s)
49  [select * from personnes where nom='Bruneau'] : Exécution réussie
50  prenom,nom,age
51  --------------
52  Josette,Bruneau,46
53  [xselect * from personnes where nom='Bruneau'] : Erreur (SQLSTATE[42000]: Syntax error or access violation:
    1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
    for the right syntax to use near 'xselect * from personnes where nom='Bruneau'' at line 1)
54  [] : Erreur (Transaction annulée)
55
56  ----------------------
57  Il y a eu 2 erreur(s)
58  [xselect * from personnes where nom='Bruneau'] : Erreur (SQLSTATE[42000]: Syntax error or access violation:
    1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
    for the right syntax to use near 'xselect * from personnes where nom='Bruneau'' at line 1)
59  [] : Erreur (Transaction annulée)
60  Terminé
```

- ligne 53 : une erreur se produit sur la commande **[xselect]** ;
- ligne 54 : la transaction est alors annulée ;

Si on vérifie l'état de la base, on la trouve dans le même état qu'avant l'exécution du script. On n'y voit pas notamment la ligne **[Josette, Bruneau, 46]** de la ligne 52 des résultats ci-dessus.

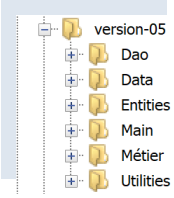


Résumé

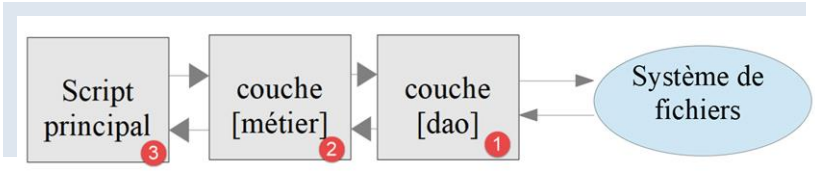
- une transaction commence avec la méthode **[PDO::beginTransaction]** ;
- on la termine sur un succès avec la méthode **[PDO::commit]** ;
- on la termine sur un échec avec la méthode **[PDO::rollback]** ;

Lorsqu'on exploite une base de données, c'est une bonne habitude de mettre toute opération SQL dans une transaction pour s'isoler des autres utilisateurs de la base (elle a également ce rôle). Une transaction doit être la plus courte possible. Il ne faut donc pas oublier de la terminer par un **[commit]** ou un **[rollback]** selon les cas.

1.13 Exercice d'application – version 5



Nous avons déjà écrit plusieurs versions de cet exercice. La dernière version utilisait une architecture en couches :



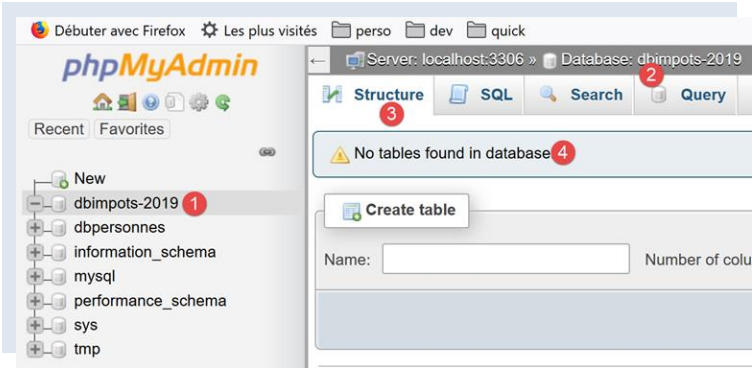
La couche **[dao]** implémente une interface **[InterfaceDao]**. Nous avons construit une classe implémentant de cette interface :

- **[DaoImpotsWithTaxAdminDataInJsonFile]** qui allait chercher les données fiscales dans un fichier jSON ;

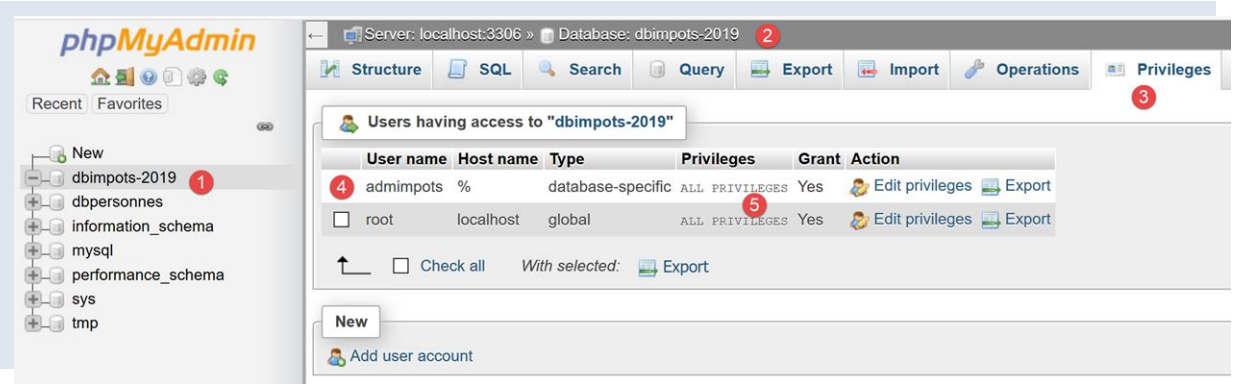
Nous allons implémenter l'interface **[InterfaceDao]** par une nouvelle classe **[DaoImpotsWithTaxAdminDataInDatabase]** qui ira chercher les données de l'administration fiscale dans une base de données MySQL.

1.13.1 Création de la base de données [dbimpots-2019]

En suivant l'exemple du paragraphe lien, nous construisons une base de données MySQL nommée **[dbimpots-2019]** dont le propriétaire sera **[admimpots]** avec le mot de passe **[mdpimpots]** :



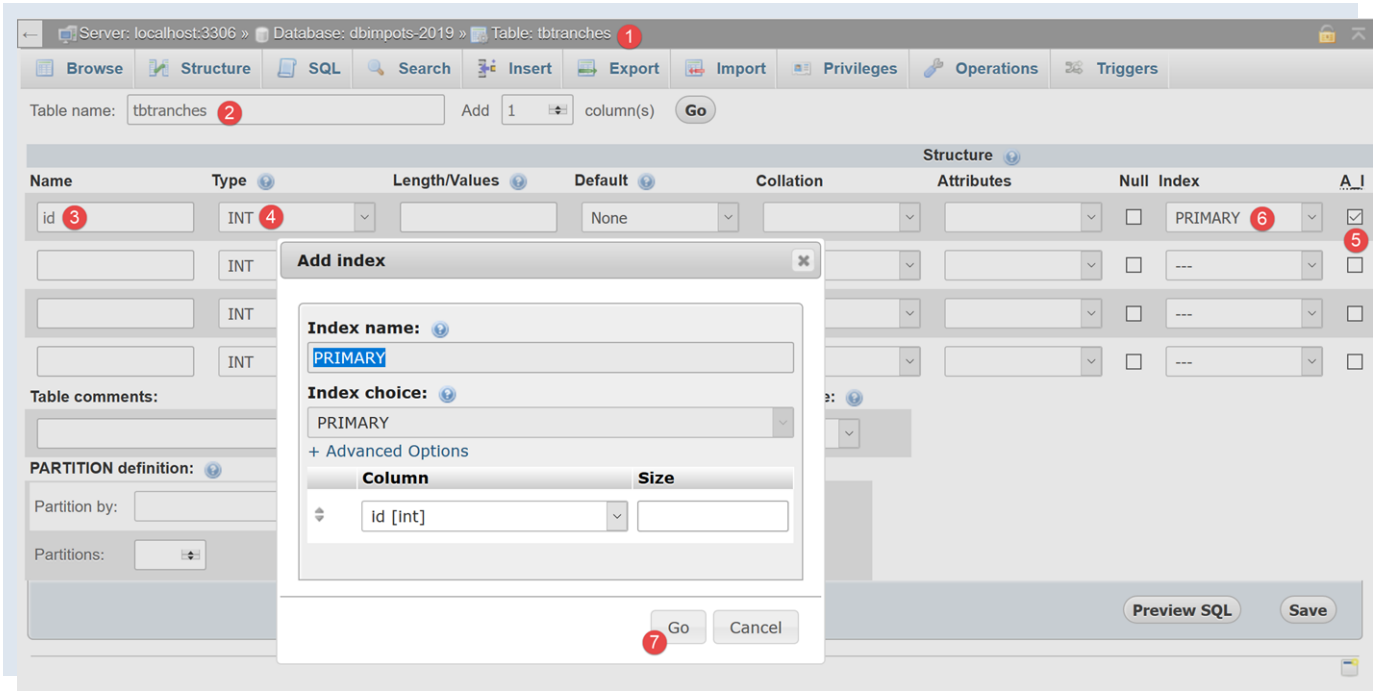
- en **[1-4]** ci-dessus, nous voyons la base **[dbimpots-2019]** qui pour l'instant n'a pas de tables ;



- en **[1-5]** ci-dessus, nous voyons que l'utilisateur **[admimpots]** a tous les droits sur la base **[dbimpots-2019]**. Ce que nous ne voyons pas ici c'est que cet utilisateur a le mot de passe **[admimpots]** ;

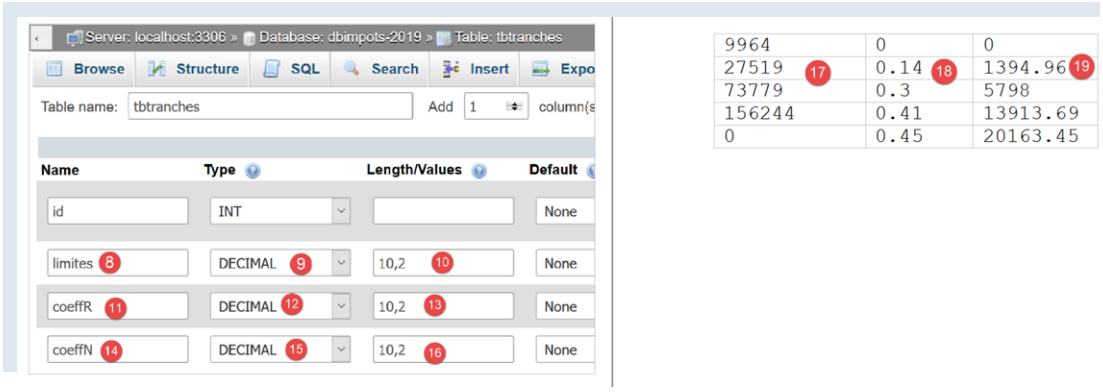
Nous créons maintenant la table **[tbtranches]**qui contiendra les tranches d'imposition :

- en **[1-7]**, nous créons une table nommée **[tbtranches]** ayant 4 colonnes ;



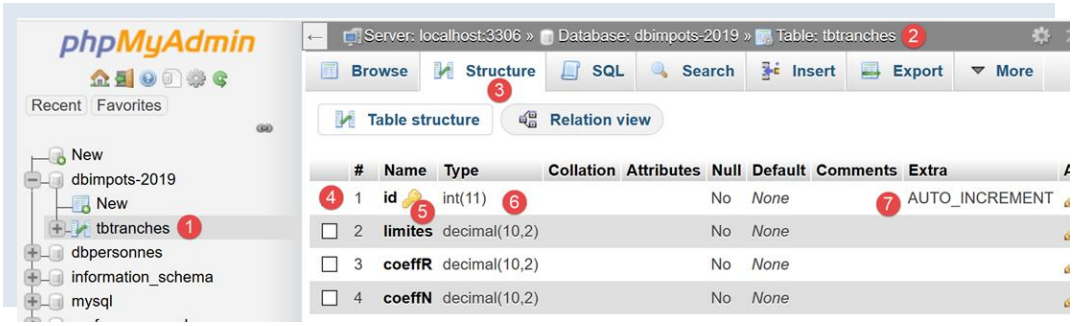
- en **[3-6]** nous définissons une colonne nommée **[id]** (3), de type entier **[int]** (4), qui sera clé primaire **[6]** de la table et sera autoincrémentée **[5]** par le SGBD. Cela signifie que MySQL va gérer lui-même les valeurs de la clé primaire au moment des insertions. Il affectera la valeur 1 à la clé primaire de la 1ière insertion, puis 2 à la suivante, etc … ;

- en **[7]**, l'assistant nous propose d'autres options de configuration de la clé primaire. Ici on se contente de valider **[7]** les valeurs par défaut ;



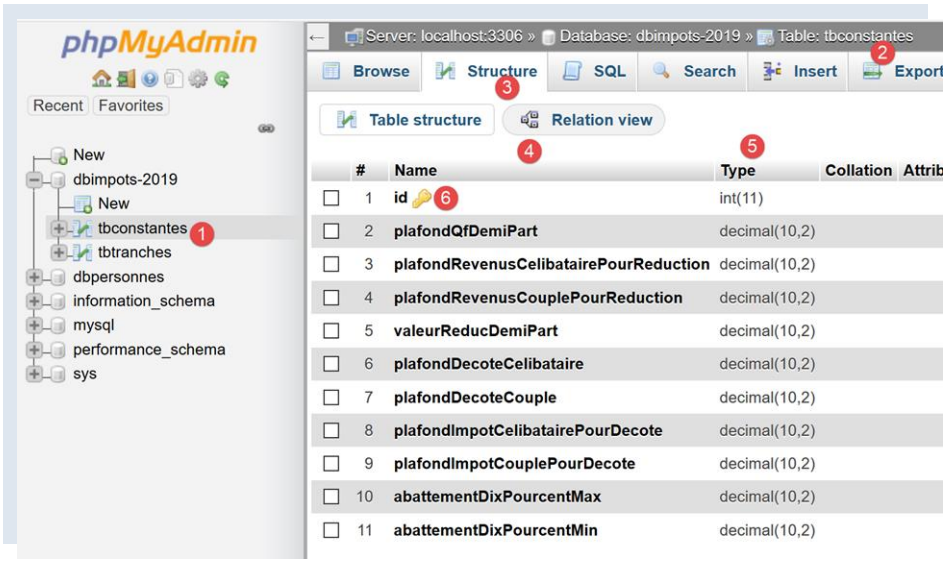
- en **[8-16]**, on définit les trois autres colonnes de la table :
  - **[limites]** (8) de type nombre décimal (9) à 10 chiffres dont 2 décimales (10) contiendra les éléments de la colonne 17 des tranches d'impôts ;
  - **[coeffR]** (11) de type nombre décimal (12) à 6 chiffres dont 2 décimales (13) contiendra les éléments de la colonne 18 des tranches d'impôts ;
  - **[coeffN]** (14) de type nombre décimal (15) à 10 chiffres dont 2 décimales (16) contiendra les éléments de la colonne 19 des tranches d'impôts ;

Après avoir validé cette structure, nous obtenons le résultat suivant :

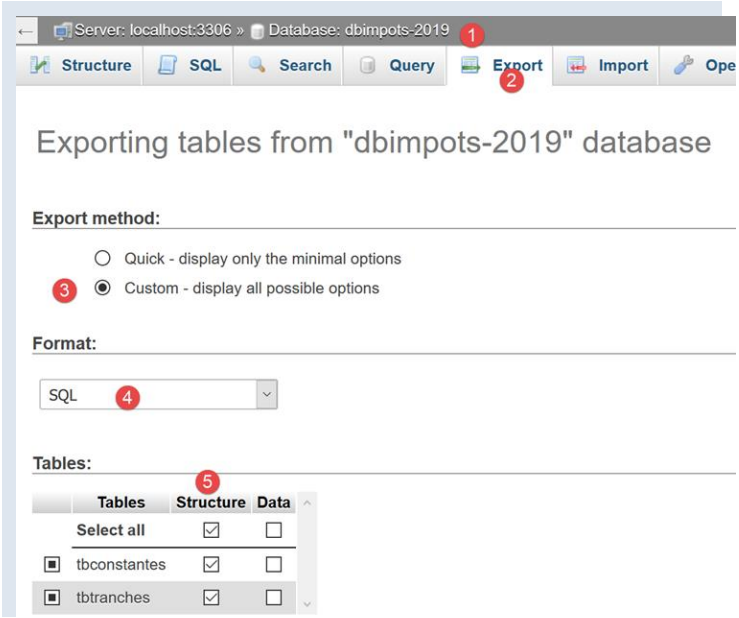


- en **[5]**, l'icône de la clé indique que la colonne **[id]** est clé primaire. On voit également que cette clé primaire a des valeurs entières (6) et qu'elle est gérée (autoincrémentée) par MySQL ;

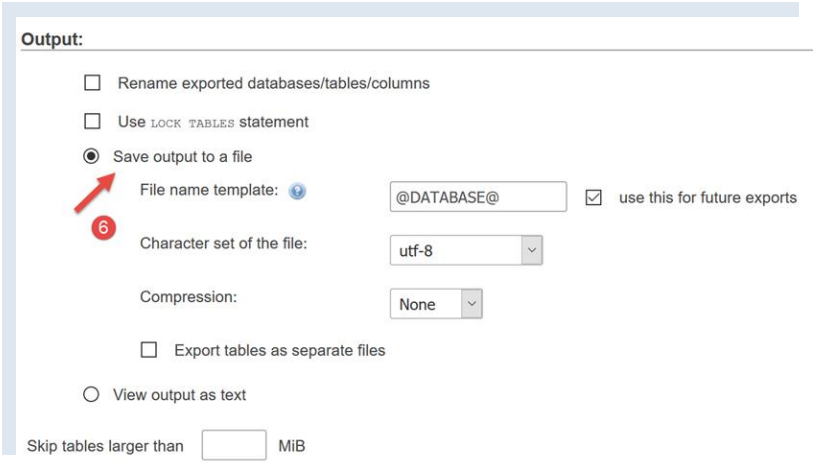
De la même façon que nous avons créé la table **[tbtranches]** nous construisons la table **[tbconstantes]** qui contiendra les constantes du calcul de l'impôt :

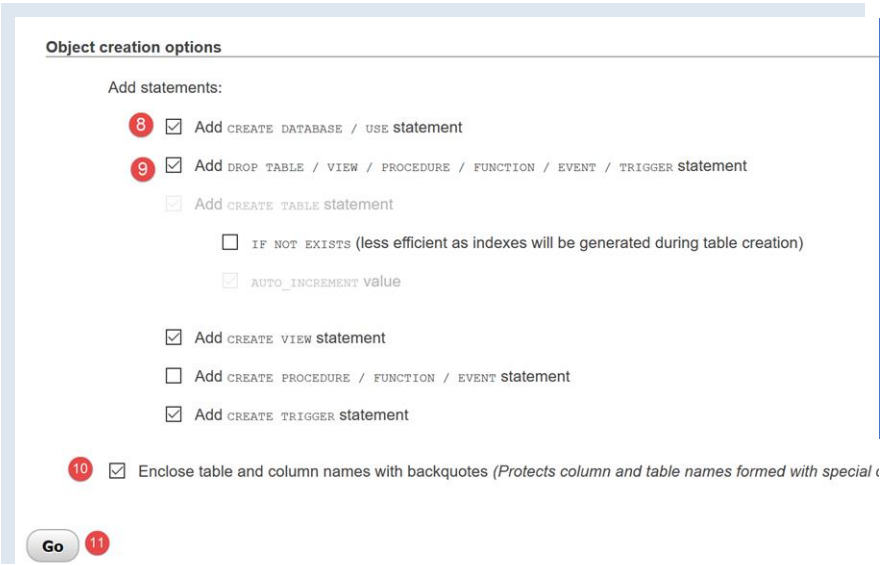


Il est possible d'exporter la structure de la base de données dans un fichier texte sous forme d'une suite d'ordres SQL :



L'option **[5]** n'exporte ici que la structure de la base de données et pas son contenu. Dans notre cas, la base n'a pas encore de contenu.



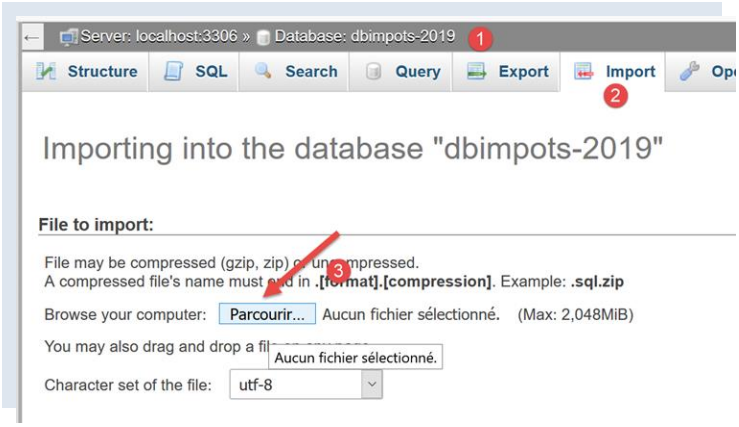


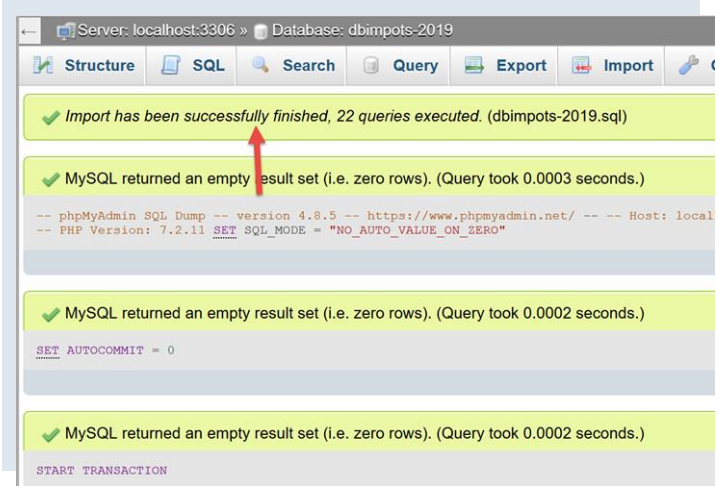
L'option **[11]** produit le fichier SQL **[dbimpots-2019.sql]** suivant :

```sql
-- phpMyAdmin SQL Dump
-- version 4.8.5
-- https://www.phpmyadmin.net/
--
-- Host: localhost:3306
-- Generation Time: Jun 30, 2019 at 01:10 PM
-- Server version: 5.7.24
-- PHP Version: 7.2.11

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `dbimpots-2019`
--
CREATE DATABASE IF NOT EXISTS `dbimpots-2019` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE `dbimpots-2019`;

-- --------------------------------------------------------

--
-- Table structure for table `tbconstantes`
--

DROP TABLE IF EXISTS `tbconstantes`;
CREATE TABLE `tbconstantes` (
  `id` int(11) NOT NULL,
  `plafondQfDemiPart` decimal(10,2) NOT NULL,
  `plafondRevenusCelibatairePourReduction` decimal(10,2) NOT NULL,
  `plafondRevenusCouplePourReduction` decimal(10,2) NOT NULL,
  `valeurReducDemiPart` decimal(10,2) NOT NULL,
  `plafondDecoteCelibataire` decimal(10,2) NOT NULL,
  `plafondDecoteCouple` decimal(10,2) NOT NULL,
  `plafondImpotCelibatairePourDecote` decimal(10,2) NOT NULL,
  `plafondImpotCouplePourDecote` decimal(10,2) NOT NULL,
  `abattementDixPourcentMax` decimal(10,2) NOT NULL,
  `abattementDixPourcentMin` decimal(10,2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `tbtranches`
--

DROP TABLE IF EXISTS `tbtranches`;
CREATE TABLE `tbtranches` (
  `id` int(11) NOT NULL,
  `limites` decimal(10,2) NOT NULL,
  `coeffR` decimal(10,2) NOT NULL,
  `coeffN` decimal(10,2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Indexes for dumped tables
--

--
-- Indexes for table `tbconstantes`
--
ALTER TABLE `tbconstantes`
  ADD PRIMARY KEY (`id`);

--
-- Indexes for table `tbtranches`
--
ALTER TABLE `tbtranches`
```

```
76    ADD PRIMARY KEY (`id`);
77
78  --
79  -- AUTO_INCREMENT for dumped tables
80  --
81
82  --
83  -- AUTO_INCREMENT for table `tbconstantes`
84  --
85  ALTER TABLE `tbconstantes`
86    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
87
88  --
89  -- AUTO_INCREMENT for table `tbtranches`
90  --
91  ALTER TABLE `tbtranches`
92    MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
93  COMMIT;
94
95  /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
96  /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
97  /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

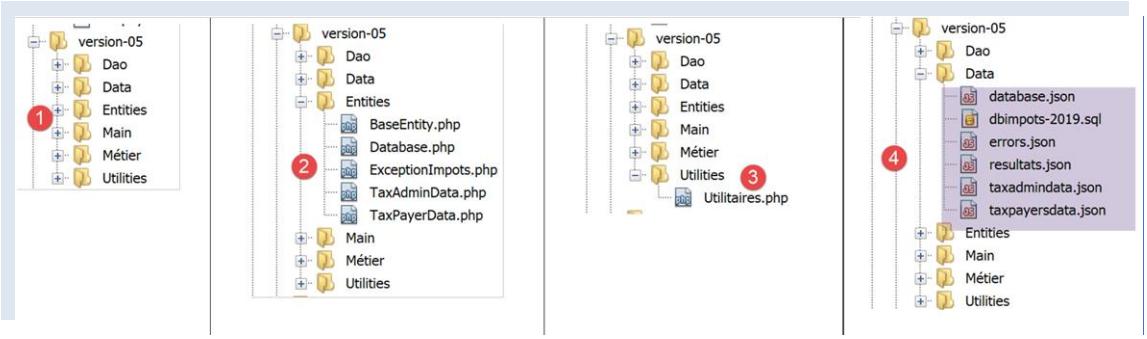
Vous pouvez utiliser ce fichier SQL pour régénérer la base **[dbimpots-2019]** si elle a été détruite ou altérée. Il n'y a pas lieu ici de supprimer la base avant de la régénérer puisque le script SQL prend soin de le faire lui-même :





1.13.2 Organisation du code

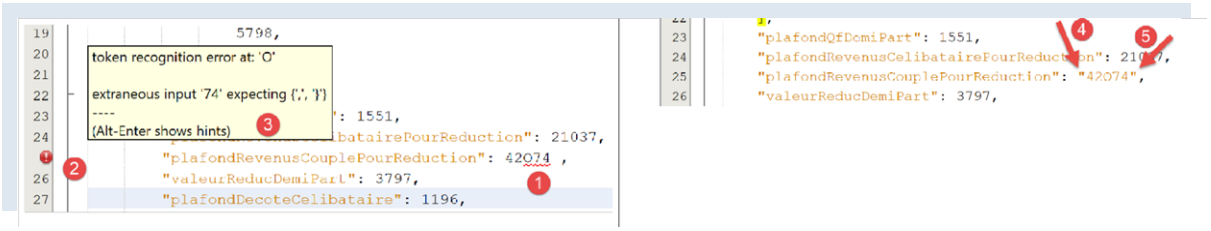
Pour mieux montrer le rôle des différents scripts PHP que nous écrivons, nous allons organiser notre code en dossiers :



- en **[1]**, vue d'ensemble de la version 05 ;
- en **[2]**, les entités de l'application, entités échangées entre couches ;
- en **[3]**, les utilitaires de l'application ;
- en **[4]**, les données utilisées ou produites par l'application. Nous prenons ici la décision de n'utiliser que des fichiers jSON pour les fichiers texte. Ceux-ci présentent plusieurs avantages :
  - ils sont reconnus par beaucoup d'outils ;
  - ces outils ont une coloration syntaxique. Par ailleurs, la notation jSON a des règles. Lorsque celles-ci ne sont pas respectées les outils les signalent. Par exemple, une erreur difficile à détecter dans un fichier texte basique est l'utilisation de O majuscule / minuscule à la place de zéros. Si cette erreur se produit elle sera signalée. En effet dans le code jSON :

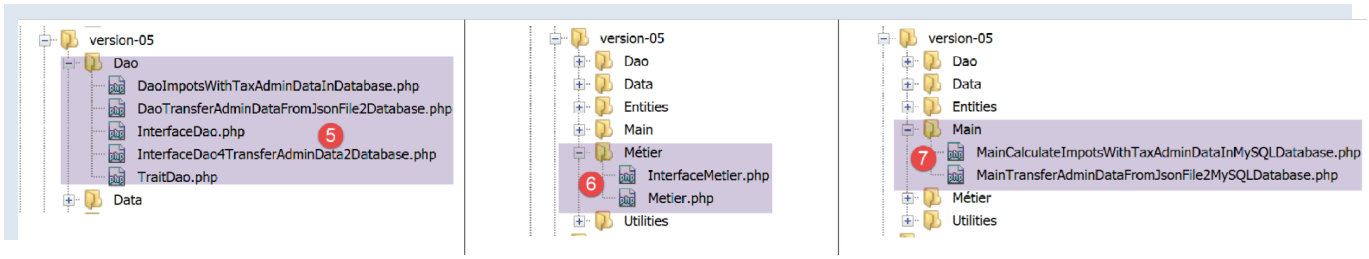
```
"plafondRevenusCouplePourReduction": 42074
```

où on a mis par inadvertance un O majuscule à la place du zéro dans **[42074]**, Netbeans signale la faute :



En effet, Netbeans reconnaît le O majuscule qui fait de **[49074]** une chaîne de caractères. Il en conclut que la syntaxe devrait être **[4-5]** : la chaîne **[47074]** devrait être entourée de guillemets. L'attention du développeur est donc attirée par la faute et peut la corriger : soit mettre les guillemets, soit remplacer le O par un zéro ;

Les autres éléments de la version 05 sont les suivants :



- en **[6]**, les interfaces et classes de la couche **[Dao]** ;
- en **[7]**, les interfaces et classes de la couche **[métier]** ;
- en **[8]**, les scripts principaux de la version 05 ;

La version 05 a deux objectifs distincts :

- remplir la base MySQL **[dbimpots-2019]** avec le contenu du fichier jSON **[Data/txadmindata.json]** ;
- implémenter le calcul de l'impôt avec des données fiscales venant désormais de la base MySQL **[dbimpots-2019]** ;

Nous allons traiter ces deux objectifs séparément.

1.13.3 Remplissage de base de données `[dbimpots-2019]`

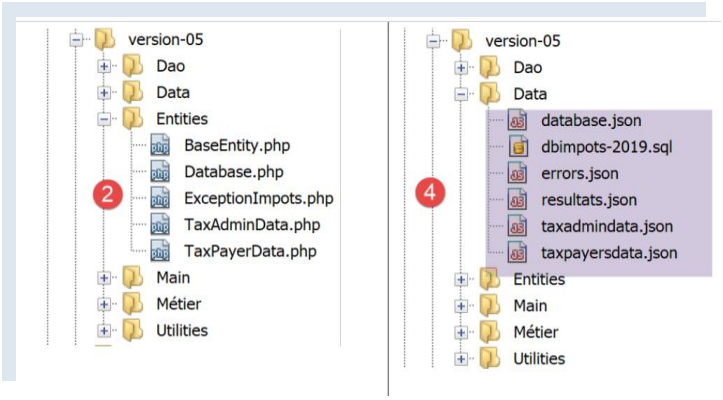
1.13.3.1 Objectif

Le fichier texte *taxadmindata.json* contient les données de l'administration fiscale :

```
{
    "limites": [
        9964,
        27519,
        73779,
        156244,
        0
    ],
    "coeffR": [
        0,
        0.14,
        0.3,
        0.41,
        0.45
    ],
    "coeffN": [
        0,
        1394.96,
        5798,
        13913.69,
        20163.45
    ],
    "plafondQfDemiPart": 1551,
    "plafondRevenusCelibatairePourReduction": 21037,
    "plafondRevenusCouplePourReduction": 42074,
    "valeurReducDemiPart": 3797,
    "plafondDecoteCelibataire": 1196,
    "plafondDecoteCouple": 1970,
    "plafondImpotCouplePourDecote": 2627,
    "plafondImpotCelibatairePourDecote": 1595,
    "abattementDixPourcentMax": 12502,
    "abattementDixPourcentMin": 437
}
```

Notre objectif est de transférer ces données dans la base MySQL **`[dbimpots-2019]`** créée précédemment.

1.13.3.2 Les entités



L'entité **`[Database]`** servira à encapsuler les données du fichier jSON **`[database.json]`** suivant :

```
{
    "dsn": "mysql:host=localhost;dbname=dbimpots-2019",
    "id": "admimpots",
    "pwd": "mdpimpots",
    "tableTranches": "tbtranches",
    "colLimites": "limites",
    "colCoeffR": "coeffr",
    "colCoeffN": "coeffn",
    "tableConstantes": "tbconstantes",
    "colPlafondQfDemiPart": "plafondQfDemiPart",
    "colPlafondRevenusCelibatairePourReduction": "plafondRevenusCelibatairePourReduction",
```

```
    "colPlafondRevenusCouplePourReduction": "plafondRevenusCouplePourReduction",
    "colValeurReducDemiPart": "valeurReducDemiPart",
    "colPlafondDecoteCelibataire": "plafondDecoteCelibataire",
    "colPlafondDecoteCouple": "plafondDecoteCouple",
    "colPlafondImpotCelibatairePourDecote": "plafondImpotCelibatairePourDecote",
    "colPlafondImpotCouplePourDecote": "plafondImpotCouplePourDecote",
    "colAbattementDixPourcentMax": "abattementDixPourcentMax",
    "colAbattementDixPourcentMin": "abattementDixPourcentMin"
}
```

L'entité **[TaxAdminData]** servira à encapsuler les données du fichier jSON **[taxadmindata.json]** suivant :

```
{
        "limites": [
                9964,
                27519,
                73779,
                156244,
                0
        ],
        "coeffR": [
                0,
                0.14,
                0.3,
                0.41,
                0.45
        ],
        "coeffN": [
                0,
                1394.96,
                5798,
                13913.69,
                20163.45
        ],
        "plafondQfDemiPart": 1551,
        "plafondRevenusCelibatairePourReduction": 21037,
        "plafondRevenusCouplePourReduction": 42074,
        "valeurReducDemiPart": 3797,
        "plafondDecoteCelibataire": 1196,
        "plafondDecoteCouple": 1970,
        "plafondImpotCouplePourDecote": 2627,
        "plafondImpotCelibatairePourDecote": 1595,
        "abattementDixPourcentMax": 12502,
        "abattementDixPourcentMin": 437
}
```

L'entité **[TaxPayerData]** servira à encapsuler les données du fichier jSON **[taxpayerdata.json]** suivant :

```
[
    {
        "marié": "oui",
        "enfants": 2,
        "salaire": 55555
    },
    {
        "marié": "ouix",
        "enfants": "2x",
        "salaire": "55555x"
    },
    {
        "marié": "oui",
        "enfants": "2",
        "salaire": 50000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 50000
    },
    {
        "marié": "non",
        "enfants": 2,
        "salaire": 100000
    },
    {
        "marié": "non",
```

```
        "enfants": 3,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 5,
        "salaire": 100000
    },
    {
        "marié": "non",
        "enfants": 0,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 2,
        "salaire": 30000
    },
    {
        "marié": "non",
        "enfants": 0,
        "salaire": 200000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 20000
    }
]
```

1.13.3.2.1 La classe de base `[BaseEntity]`

Pour simplifier le code des entités, nous adopterons la règle suivante : **les attributs d'une entité ont les mêmes noms que les attributs du fichier jSON que l'entité doit encapsuler**. Moyennant cette règle, les entités `[Database, TaxAdminData, TaxPayerData]` ont des points communs qui peuvent être factorisés dans une classe parent. Ce sera la classe `[BaseEntity]` suivante :

```
<?php

namespace Application;

class BaseEntity {
  // attribut
  protected $arrayOfAttributes;

  // initialisation à partir d'un fichier jSON
  public function setFromJsonFile(string $jsonFilename) {
    // on récupère le contenu du fichier des données fiscales
    $fileContents = \file_get_contents($jsonFilename);
    $erreur = FALSE;
    // erreur ?
    if (!$fileContents) {
      // on note l'erreur
      $erreur = TRUE;
      $message = "Le fichier des données [$jsonFilename] n'existe pas";
    }
    if (!$erreur) {
      // on récupère le code jSON du fichier de configuration dans un tableau associatif
      $this->arrayOfAttributes = \json_decode($fileContents, true);
      // erreur ?
      if ($this->arrayOfAttributes === FALSE) {
        // on note l'erreur
        $erreur = TRUE;
        $message = "Le fichier de données jSON [$jsonFilename] n'a pu être exploité correctement";
      }
    }
    // erreur ?
    if ($erreur) {
      // on lance une exception
      throw new ExceptionImpots($message);
    }
    // initialisation des attributs de la classe
    foreach ($this->arrayOfAttributes as $key => $value) {
```

```
37        $this->$key = $value;
38      }
39      // on rend l'objet
40      return $this;
41    }
42
43    public function checkForAllAttributes() {
44      // on vérifie que toutes les clés ont été initialisées
45      foreach (\array_keys($this->arrayOfAttributes) as $key) {
46        if ($key !== "arrayOfAttributes" && !isset($this->$key)) {
47          throw new ExceptionImpots("L'attribut [$key] de la classe "
48            . get_class($this) . " n'a pas été initialisé");
49        }
50      }
51    }
52
53    public function setFromArrayOfAttributes(array $arrayOfAttributes) {
54      // on initialise certains attributs de la classe
55      foreach ($arrayOfAttributes as $key => $value) {
56        $this->$key = $value;
57      }
58      // on retourne l'objet
59      return $this;
60    }
61
62    // toString
63    public function __toString() {
64      // attributs de l'objet
65      $arrayOfAttributes = \get_object_vars($this);
66      // on enlève l'attribut de la classe parent
67      unset($arrayOfAttributes["arrayOfAttributes"]);
68      // chaîne Json de l'objet
69      return \json_encode($arrayOfAttributes, JSON_UNESCAPED_UNICODE);
70    }
71
72    // getter
73    public function getArrayOfAttributes() {
74      return $this->arrayOfAttributes;
75    }
76
77  }
```

Commentaires

- ligne 5 : la classe **[BaseEntity]** est destinée à être étendue par les classes **[Database, TaxAdminData, TaxPayerData]** ;
- ligne 7 : l'attribut **[$arrayOfAttributes]** est un tableau contenant tous les attributs de la classe fille ayant étendu **[BaseEntity]** ainsi que leurs valeurs ;
- lignes 9-41 : l'attribut **[$arrayOfAttributes]** est initialisé à partir du fichier jSON **[$jsonFilename]** passé en paramètre. Une exception de type **[ExceptionImpot]** est lancée si le fichier jSON n'a pu être lu ou si ce n'est pas un fichier jSON valide ;
- lignes 36-38 : on a là un code spécial s'il est exécuté par une classe fille. Dans ce cas, **[$this]** représente une instance de la classe fille **[Database, TaxAdminData, TaxPayerData]** et dans ce cas là, **les lignes 36-38 initialisent les attributs de cette classe fille,** à condition que ces attributs aient la visibilité **protected** (ou public) (cf paragraphe lien). On a dit en effet que les attributs des entités **[Database, TaxAdminData, TaxPayerData]** étaient les mêmes que les attributs du fichier jSON qu'ils encapsulaient. Finalement, la méthode **[setFromJsonFile]** permet à une classe fille de s'initialiser à partir d'un fichier jSON ;
- ligne 40 : on rend l'objet **[$this]** donc une instance de classe fille si la méthode **[setFromJsonFile]** a été appelée par une classe fille ;
- lignes 43-51 : la méthode **[checkForAllAttributes]** permet à une classe fille de vérifier que tous ses attributs ont été initialisés. Si ce n'est pas le cas, une exception **[ExceptionImpots]** est lancée. Cette méthode permet à la classe fille de vérifier que son fichier jSON n'a pas oublié certains attributs ;
- lignes 53-60 : la méthode **[setFromArrayOfAttributes]** permet à une classe fille d'initialiser tout ou partie de ses attributs à partir d'un tableau associatif dont les clés ont les mêmes noms que les attributs de la classe fille à initialiser ;
- lignes 63-70 : la méthode **[__toString]** permet d'avoir la représentation jSON d'une classe fille ;

1.13.3.2.2 L'entité [Database]

L'entité **[Database]** est la suivante :

```
1   <?php
2
3   namespace Application;
4
5   class Database extends BaseEntity {
6     // attributs
```

```
7     protected $dsn;
8     protected $id;
9     protected $pwd;
10    protected $tableTranches;
11    protected $colLimites;
12    protected $colCoeffR;
13    protected $colCoeffN;
14    protected $tableConstantes;
15    protected $colPlafondQfDemiPart;
16    protected $colPlafondRevenusCelibatairePourReduction;
17    protected $colPlafondRevenusCouplePourReduction;
18    protected $colValeurReducDemiPart;
19    protected $colPlafondDecoteCelibataire;
20    protected $colPlafondDecoteCouple;
21    protected $colPlafondImpotCelibatairePourDecote;
22    protected $colPlafondImpotCouplePourDecote;
23    protected $colAbattementDixPourcentMax;
24    protected $colAbattementDixPourcentMin;
25
26   …
27
28  }
```

La classe **[Database]** est utilisée pour encapsuler les données du fichier jSON **[database.json]** suivant :

```
1   {
2       "dsn": "mysql:host=localhost;dbname=dbimpots-2019",
3       "id": "admimpots",
4       "pwd": "mdpimpots",
5       "tableTranches": "tbtranches",
6       "colLimites": "limites",
7       "colCoeffR": "coeffr",
8       "colCoeffN": "coeffn",
9       "tableConstantes": "tbconstantes",
10      "colPlafondQfDemiPart": "plafondQfDemiPart",
11      "colPlafondRevenusCelibatairePourReduction": "plafondRevenusCelibatairePourReduction",
12      "colPlafondRevenusCouplePourReduction": "plafondRevenusCouplePourReduction",
13      "colValeurReducDemiPart": "valeurReducDemiPart",
14      "colPlafondDecoteCelibataire": "plafondDecoteCelibataire",
15      "colPlafondDecoteCouple": "plafondDecoteCouple",
16      "colPlafondImpotCelibatairePourDecote": "plafondImpotCelibatairePourDecote",
17      "colPlafondImpotCouplePourDecote": "plafondImpotCouplePourDecote",
18      "colAbattementDixPourcentMax": "abattementDixPourcentMax",
19      "colAbattementDixPourcentMin": "abattementDixPourcentMin"
20  }
```

La classe et le fichier jSON ont les mêmes attributs. Ceux-ci décrivent les caractéristiques de la base de données MySQL **[dbimpots-2019]** :

`dsn`	Nom DSN de la base
`id`	Propriétaire de la base
`pwd`	Son mot de passe
`tableTranches`	Nom de la table contenant les tranches d'imposition

`colLimites`
`colCoeffR`
`colCoeffN`
	Noms des colonnes de la table **[tableTranches]**

`tableConstantes`	Nom de la table contenant les constantes de calcul de l'impôt

`colPlafondQfDemiPart`
`colPlafondRevenusCelibatairePourReduction`
`colPlafondRevenusCouplePourReduction`
`colValeurReducDemiPart`
`colPlafondDecoteCelibataire`
`colPlafondDecoteCouple`
`colPlafondImpotCelibatairePourDecote`
`colPlafondImpotCouplePourDecote`
`colAbattementDixPourcentMax`
`colAbattementDixPourcentMin`
	Noms des colonnes de la table **[tableConstantes]** contenant les constantes de calcul de l'impôt

Pourquoi nommer les tables et les colonnes alors qu'on connaît déjà leurs noms et que ce n'est pas quelque chose amené à changer ? Après le SGBD MySQL, on va utiliser le SGBD PostgreSQL pour stocker les données de l'administration fiscale. Or les noms des colonnes et tables Postgres ne suivent pas les mêmes règles que MySQL. On va être obligés d'utiliser d'autres noms. C'est également

vrai pour d'autres SGBD. Si on veut avoir du code portable entre SGBD, il est alors préférable d'utiliser des paramètres plutôt que les noms en dur des tables et colonnes.

Revenons au code de la classe **[Database]** :

```
1   <?php
2
3   namespace Application;
4
5   class Database extends BaseEntity {
6     // attributs
7     protected $dsn;
8     protected $id;
9     protected $pwd;
10    protected $tableTranches;
11    protected $colLimites;
12    protected $colCoeffR;
13    protected $colCoeffN;
14    protected $tableConstantes;
15    protected $colPlafondQfDemiPart;
16    protected $colPlafondRevenusCelibataire PourReduction;
17    protected $colPlafondRevenusCouplePourReduction;
18    protected $colValeurReducDemiPart;
19    protected $colPlafondDecoteCelibataire;
20    protected $colPlafondDecoteCouple;
21    protected $colPlafondImpotCelibataire PourDecote;
22    protected $colPlafondImpotCouplePourDecote;
23    protected $colAbattementDixPourcentMax;
24    protected $colAbattementDixPourcentMin;
25
26    // setter
27    // initialisation
28    public function setFromJsonFile(string $jsonFilename) {
29      // parent
30      parent::setFromJsonFile($jsonFilename);
31      // on vérifie que tous les attributs ont été initialisés
32      parent::checkForAllAttributes();
33      // on retourne l'objet
34      return $this;
35    }
36
37    // getters et setters
38    public function getDsn() {
39      return $this->dsn;
40    }
41
42    …
43
44    public function setDsn($dsn) {
45      $this->dsn = $dsn;
46      return $this;
47    }
48
49    …
50
51  }
```

Commentaires

- lignes 7-24 : tous les attributs de la classe ont la visibilité **[protected]**. C'est la condition pour qu'ils puissent être modifiés depuis la classe parent **[BaseEntity]** (cf paragraphe lien) ;
- lignes 28-35 : la méthode **[setFromJsonFile]** permet d'initialiser les attributs de la classe **[Database]** à partir du contenu d'un fichier jSON passé en paramètre. Il faut que les attributs du fichier jSON et ceux de la classe **[Database]** soient identiques. Si le fichier jSON n'est pas exploitable, une exception est lancée ;
- ligne 30 : c'est la classe parent qui fait l'initialisation ;
- ligne 32 : on demande à la classe parent de vérifier que tous les attributs de la classe **[Database]** ont été initialisés. Si ce n'est pas le cas, une exception est lancée ;
- ligne 34 : on rend l'instance **[Database]** qui vient d'être initialisée ;
- lignes 37 et au-delà : les getters et setters des attributs de la classe ;

1.13.3.2.3 L'entité [TaxAdminData]

L'entité **[TaxAdminData]** est la suivante :

```php
<?php

namespace Application;

class TaxAdminData extends BaseEntity {
  // tranches d'impôt
  protected $limites;
  protected $coeffR;
  protected $coeffN;
  // constantes de calcul de l'impôt
  protected $plafondQfDemiPart;
  protected $plafondRevenusCelibataire PourReduction;
  protected $plafondRevenusCouplePourReduction;
  protected $valeurReducDemiPart;
  protected $plafondDecoteCelibataire;
  protected $plafondDecoteCouple;
  protected $plafondImpotCouplePourDecote;
  protected $plafondImpotCelibatairePourDecote;
  protected $abattementDixPourcentMax;
  protected $abattementDixPourcentMin;

  …
}
```

La classe **[TaxAdminData]** est utilisée pour encapsuler les données du fichier jSON **[taxadmindata.json]** suivant :

```json
{
    "limites": [
        9964,
        27519,
        73779,
        156244,
        0
    ],
    "coeffR": [
        0,
        0.14,
        0.3,
        0.41,
        0.45
    ],
    "coeffN": [
        0,
        1394.96,
        5798,
        13913.69,
        20163.45
    ],
    "plafondQfDemiPart": 1551,
    "plafondRevenusCelibatairePourReduction": 21037,
    "plafondRevenusCouplePourReduction": 42074,
    "valeurReducDemiPart": 3797,
    "plafondDecoteCelibataire": 1196,
    "plafondDecoteCouple": 1970,
    "plafondImpotCouplePourDecote": 2627,
    "plafondImpotCelibatairePourDecote": 1595,
    "abattementDixPourcentMax": 12502,
    "abattementDixPourcentMin": 437
}
```

La classe et le fichier jSON ont les mêmes attributs. Ceux-ci représentent les données de l'administration fiscale. Le reste du code de la classe **[TaxAdminData]** est le suivant :

```php
<?php

namespace Application;

class TaxAdminData extends BaseEntity {
  // tranches d'impôt
  protected $limites;
  protected $coeffR;
  protected $coeffN;
  // constantes de calcul de l'impôt
  protected $plafondQfDemiPart;
  protected $plafondRevenusCelibataire PourReduction;
```

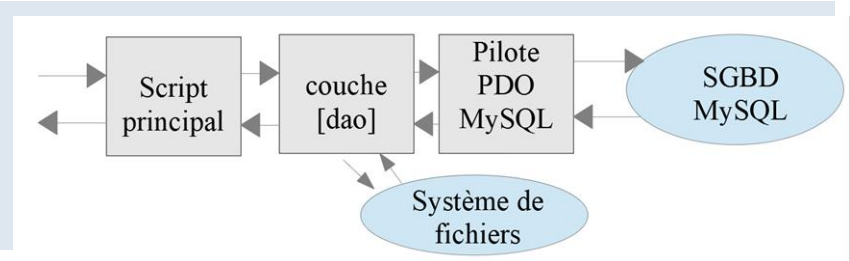
```php
  protected $plafondRevenusCouplePourReduction;
  protected $valeurReducDemiPart;
  protected $plafondDecoteCelibataire;
  protected $plafondDecoteCouple;
  protected $plafondImpotCouplePourDecote;
  protected $plafondImpotCelibatairePourDecote;
  protected $abattementDixPourcentMax;
  protected $abattementDixPourcentMin;

  // initialisation
  public function setFromJsonFile(string $taxAdminDataFilename) {
    // parent
    parent::setFromJsonFile($taxAdminDataFilename);
    // on vérifie que tous les attributs ont été initialisés
    parent::checkForAllAttributes();
    // on vérifie que les valeurs des attributs sont des réels >=0
    foreach ($this as $key => $value) {
      if ($key !== "arrayOfAttributes") {
        // $value doit être un nbre réel >=0 ou un tableau de réels >=0
        $result = $this->check($value);
        // erreur ?
        if ($result->erreur) {
          // on lance une exception
          throw new ExceptionImpots("La valeur de l'attribut [$key] est invalide");
        } else {
          // on note la valeur
          $this->$key = $result->value;
        }
      }
    }
    // on rend l'objet
    return $this;
  }

  protected function check($value): \stdClass {
    // $value est un tableau d'éléments de type string ou un unique élément
    if (!\is_array($value)) {
      $tableau = [$value];
    } else {
      $tableau = $value;
    }
    // on transforme le tableau de strings en tableau de réels
    $newTableau = [];
    $result = new \stdClass();
    // les éléments du tableau doivent être des nombres décimaux positifs ou nuls
    $modèle = '/^\s*([+]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/';
    for ($i = 0; $i < count($tableau); $i ++) {
      if (preg_match($modèle, $tableau[$i])) {
        // on met le float dans newTableau
        $newTableau[] = (float) $tableau[$i];
      } else {
        // on note l'erreur
        $result->erreur = TRUE;
        // on quitte
        return $result;
      }
    }
    // on rend le résultat
    $result->erreur = FALSE;
    if (!\is_array($value)) {
      // une seule valeur
      $result->value = $newTableau[0];
    } else {
      // une liste de valeurs
      $result->value = $newTableau;
    }
    return $result;
  }

  // getters et setters
 …
}
```

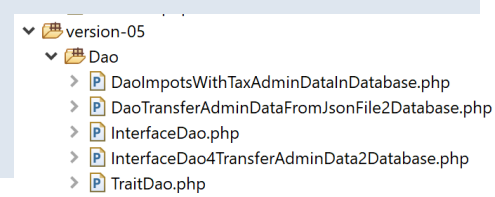
Commentaires

- ligne 23 : la méthode **[setFromJsonFile]** sert à initialiser les attributs de la classe **[TaxAdminData]** à partir d'un fichier jSON passé en paramètre. Il faut que les attributs du fichier jSON existent sous le même nom dans la classe ;
- ligne 25 : c'est la classe parent qui fait ce travail ;
- ligne 27 : on demande à la classe parent de vérifier que tous les attributs de la classe fille ont été initialisés ;
- lignes 29-42 : on vérifie localement que tous les attributs ont eu une valeur réelle positive ou nulle. Cette vérification a déjà été discutée au paragraphe lien de la version 03 ;

1.13.3.3 La couche [dao]

Maintenant nous pouvons écrire le code qui va transférer les données du fichier texte **[taxadmindata.json]** dans les tables **[tbtranches, tbconstantes]** de la base MySQL **[dbimpots-2019]**. Nous adopterons l'architecture suivante :





La couche **[dao]** implémentera l'interface **[InterfaceDao4TransferAdminDataFromFile2Database]** suivante :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   interface InterfaceDao4TransferAdminData2Database {
7
8     public function transferAdminData2Database(): void;
9   }
```

Commentaires

- ligne 8 : la méthode **[transferAdminData2Database]** a pour rôle de stocker les données de l'administration fiscale dans une base de données ;

L'interface **[InterfaceDao4TransferAdminData2Database]** sera implémentée par la classe **[DaoTransferAdminDataFromJsonFile2Database]** suivante :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   // définition d'une classe TransferAdminDataFromFile2DatabaseDao
7   class DaoTransferAdminDataFromJsonFile2Database implements InterfaceDao4TransferAdminData2Database {
8     // attributs de la base de données cible
9     private $database;
10    // données de l'administration fiscale
11    private $taxAdminData;
12
13    // constructeur
14    public function __construct(string $databaseFilename, string $taxAdminDataFilename) {
15      // on mémorise la configuration de la bd
16      $this->database = (new Database())->setFromJsonFile($databaseFilename);
17      // on mémorise les données fiscales
18      $this->taxAdminData = (new TaxAdminData())->setFromJsonFile($taxAdminDataFilename);
19    }
```

```
20
21    // transfère les données des tranches d'impôts d'un fichier texte
22    // vers la base de données
23    public function transferAdminData2Database(): void {
24      // on travaille sur la base
25      $database = $this->database;
26      try {
27        // on ouvre la connexion à la base de données
28        $connexion = new \PDO($database->getDsn(), $database->getId(), $database->getPwd());
29        // on veut qu'à chaque erreur de SGBD, une exception soit lancée
30        $connexion->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
31        // on démarre une transaction
32        $connexion->beginTransaction();
33        // on remplit la table des tranches d'impôt
34        $this->fillTableTranches($connexion);
35        // on remplit la table des constantes
36        $this->fillTableConstantes($connexion);
37        // on termine la transaction sur un succès
38        $connexion->commit();
39      } catch (\PDOException $ex) {
40        // y-a-t-il une transaction en cours ?
41        if (isset($connexion) && $connexion->inTransaction()) {
42          // on termine la transaction sur un échec
43          $connexion->rollBack();
44        }
45        // on remonte l'exception au code appelant
46        throw new ExceptionImpots($ex->getMessage());
47      } finally {
48        // on ferme la connexion
49        $connexion = NULL;
50      }
51    }
52
53
54    // remplissage de la table des tranches d'impôt
55    private function fillTableTranches($connexion): void {
56      …
57    }
58
59    // remplissage de la table des constantes
60    private function fillTableConstantes($connexion): void {
61      …
62    }
63
64  }
```

Commentaires

Nous utilisons ici ce que nous avons appris dans le chapitre sur MySQL.

- ligne 7 : la classe **[DaoTransferAdminDataFromJsonFile2Database]** implémente l'interface **[InterfaceDao4TransferAdminData2Database]** ;
- ligne 9 : l'attribut **[$database]** est l'objet de type **[Database]** encapsulant les données du fichier **[database.json]** ;
- ligne 11 : l'attribut **[$taxAdminData]** est l'objet de type **[TaxAdminData]** encapsulant les données du fichier **[taxadmindata.json]** ;
- lignes 14-19 : le constructeur reçoit en paramètres les noms des fichiers **[database.json, taxadmindata.json]** ;
- ligne 16 : initialisation de l'attribut **[$database]** ;
- ligne 18 : initialisation de l'attribut **[$taxAdminData]** ;
- ligne 23 : on implémente l'unique méthode de l'interface **[InterfaceDao4TransferAdminData2Database]** ;
- lignes 26-38 : on remplit les tables **[tbtranches, tbconstantes]** en deux temps :
  - ligne 34 : on remplit d'abord la table **[tbtranches]**. Cela se fait au sein d'une transaction (lignes 32, 38). La méthode **[fillTableTranches]** (ligne 55) lance une exception dès que quelque chose se passe mal. Dans ce cas, l'exécution se poursuit avec le catch / finally des lignes 39-50 ;
  - ligne 36 : on remplit la table **[tbconstantes]** de la même façon à l'aide de la méthode **[fillTableConstantes]** (ligne 60) ;
- lignes 39-47 : cas où une exception a été lancée par le code ;
- lignes 41-44 : si une transaction existe, elle est annulée ;
- ligne 46 : on lance une exception de type **[ExceptionImpots]** avec le message de l'exception originelle qui est, elle, d'un type quelconque ;
- lignes 47-50 : dans la clause **[finally]**, la connexion est fermée ;

Le code de la méthode **[fillTableTranches]** est le suivant :

```
1   private function fillTableTranches($connexion): void {
2       // raccourci pour la bd
3       $database = $this->database;
4       // les données à insérer dans la base de données
5       $limites = $this->taxAdminData->getLimites();
6       $coeffR = $this->taxAdminData->getCoeffR();
7       $coeffN = $this->taxAdminData->getCoeffN();
8       // on vide la table au cas où il y aurait qq chose dedans
9       $statement = $connexion->prepare("delete from " . $database->getTableTranches());
10      $statement->execute();
11      // on prépare les insertions
12      $sqlInsert = "insert into {$database->getTableTranches()} "
13        . "({$database->getColLimites()}, {$database->getColCoeffR()},"
14        . " {$database->getColCoeffN()}) values (:limites, :coeffR, :coeffN)";
15      $statement = $connexion->prepare($sqlInsert);
16      // on exécute l'ordre préparé avec les valeurs des tranches d'impôts
17      for ($i = 0; $i < count($limites); $i++) {
18        $statement->execute([
19          "limites" => $limites[$i],
20          "coeffR" => $coeffR[$i],
21          "coeffN" => $coeffN[$i]]);
22      }
23    }
```

Commentaires

- ligne 1 : la méthode **[fillTableTranches]** reçoit en paramètre une connexion ouverte. On sait de plus qu'une transaction a démarré au sein de cette connexion ;
- lignes 5-7 : les valeurs à insérer dans la table sont fournies par l'attribut **[$taxAdminData]** ;
- lignes 9-10 : on supprime le contenu actuel de la table **[tbtranches]** ;
- lignes 12-15 : on prépare l'insertion de lignes dans la table. On utilise ici les noms des colonnes fournis par l'attribut **[$database]** ;
- lignes 17-22 : on exécute autant de fois que nécessaire, l'instruction d'insertion préparée aux lignes 12-15 ;

Le code de la méthode **[fillTableConstantes]** est le suivant :

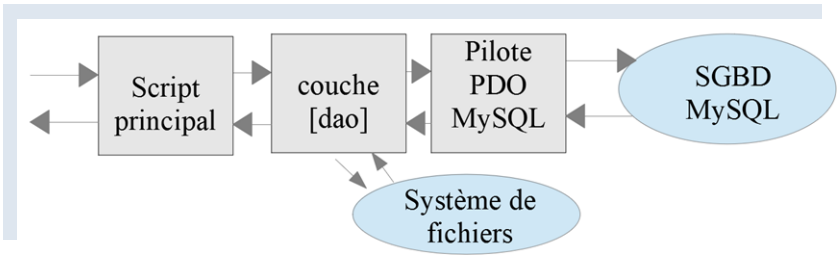
```
1   private function fillTableConstantes($connexion): void {
2       // raccourci
3       $database = $this->database;
4       // on vide la table au cas où il y aurait qq chose dedans
5       $statement = $connexion->prepare("delete from {$database->getTableConstantes()}");
6       $statement->execute();
7       // on prépare l'insertion
8       $taxAdminData = $this->taxAdminData;
9       $sqlInsert = "insert into {$database->getTableConstantes()}"
10        . " ({$database->getColPlafondQfDemiPart()},"
11        . " {$database->getColPlafondRevenusCelibatairePourReduction()},"
12        . " {$database->getColPlafondRevenusCouplePourReduction()},"
13        . " {$database->getColValeurReducDemiPart()},"
14        . " {$database->getColPlafondDecoteCelibataire()},"
15        . " {$database->getColPlafondDecoteCouple()},"
16        . " {$database->getColPlafondImpotCelibatairePourDecote()},"
17        . " {$database->getColPlafondImpotCouplePourDecote()},"
18        . " {$database->getColAbattementDixPourcentMax()},"
19        . " {$database->getColAbattementDixPourcentMin()})"
20        . " values ("
21        . ":plafondQfDemiPart,"
22        . ":plafondRevenusCelibatairePourReduction,"
23        . ":plafondRevenusCouplePourReduction,"
24        . ":valeurReducDemiPart,"
25        . ":plafondDecoteCelibataire,"
26        . ":plafondDecoteCouple,"
27        . ":plafondImpotCelibatairePourDecote,"
28        . ":plafondImpotCouplePourDecote,"
29        . ":abattementDixPourcentMax,"
30        . ":abattementDixPourcentMin)";
31      $statement = $connexion->prepare($sqlInsert);
32      // on exécute l'ordre préparé
33      $statement->execute([
34        "plafondQfDemiPart" => $taxAdminData->getPlafondQfDemiPart(),
35        "plafondRevenusCelibatairePourReduction" =>
    $taxAdminData->getPlafondRevenusCelibatairePourReduction(),
36        "plafondRevenusCouplePourReduction" => $taxAdminData->getPlafondRevenusCouplePourReduction(),
37        "valeurReducDemiPart" => $taxAdminData->getValeurReducDemiPart(),
38        "plafondDecoteCelibataire" => $taxAdminData->getPlafondDecoteCelibataire(),
```

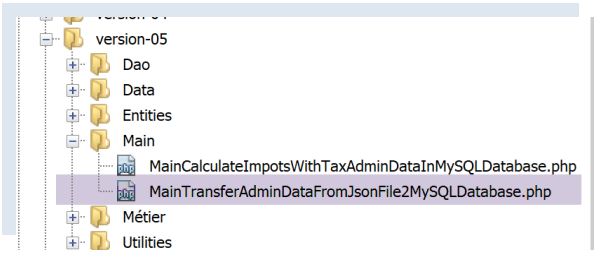
```
39          "plafondDecoteCouple" => $taxAdminData->getPlafondDecoteCouple(),
40          "plafondImpotCelibatairePourDecote" => $taxAdminData->getPlafondImpotCelibatairePourDecote(),
41          "plafondImpotCouplePourDecote" => $taxAdminData->getPlafondImpotCouplePourDecote(),
42          "abattementDixPourcentMax" => $taxAdminData->getAbattementDixPourcentMax(),
43          "abattementDixPourcentMin" => $taxAdminData->getAbattementDixPourcentMin()
44      ]);
45  }
```

Commentaires

- ligne 1 : la méthode **[fillTableConstantes]** reçoit en paramètre une connexion ouverte. On sait de plus qu'une transaction a démarré au sein de cette connexion ;
- lignes 5-6 : la table **[tbconstantes]** est vidée ;
- lignes 9-31 : préparation de l'ordre SQL d'insertion. Il est complexe du fait qu'il y a 10 colonnes à initialiser dans cette opération d'insertion et qu'il faut aller chercher les noms des colonnes dans l'attribut **[$database]** ;
- ligne 33-44 : exécution de l'ordre d'insertion. Il n'y a qu'une ligne à insérer. Là encore, le code est rendu complexe du fait qu'il faille chercher les valeurs à insérer dans l'attribut **[$taxAdminData]** ;

1.13.3.4 Le script principal





Le script principal s'appuie sur la couche **[dao]** pour opérer le transfert de données :

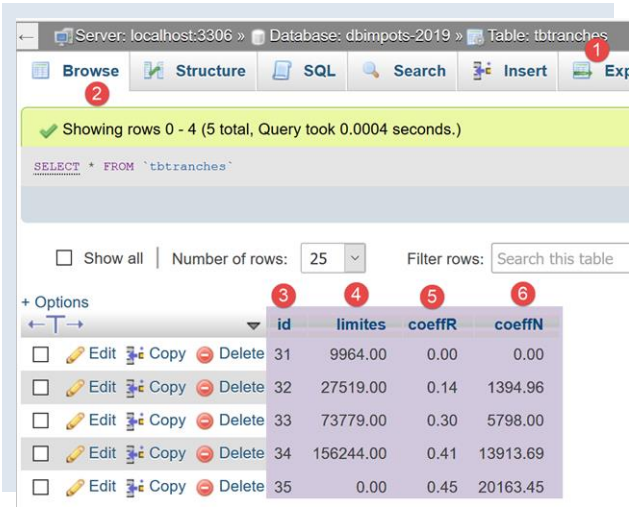
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // gestion des erreurs par PHP
10  // ini_set("display_errors", "0");
11  // inclusion interface et classes
12  require_once __DIR__ . "/../Entities/BaseEntity.php";
13  require_once __DIR__ . "/../Entities/TaxAdminData.php";
14  require_once __DIR__ . "/../Entities/TaxPayerData.php";
15  require_once __DIR__ . "/../Entities/Database.php";
16  require_once __DIR__ . "/../Entities/ExceptionImpots.php";
17  require_once __DIR__ . "/../Utilities/Utilitaires.php";
18  require_once __DIR__ . "/../Dao/InterfaceDao.php";
19  require_once __DIR__ . "/../Dao/TraitDao.php";
20  require_once __DIR__ . "/../Dao/InterfaceDao4TransferAdminData2Database.php";
21  require_once __DIR__ . "/../Dao/DaoTransferAdminDataFromJsonFile2Database.php";
22  //
23  // définition des constantes
24  const DATABASE_CONFIG_FILENAME = "../Data/database.json";
25  const TAXADMINDATA_FILENAME = "../Data/taxadmindata.json";
26
27  //
28  try {
```

```
29    // création de la couche [dao]
30    $dao = new DaoTransferAdminDataFromJsonFile2Database(DATABASE_CONFIG_FILENAME, TAXADMINDATA_FILENAME);
31    // transfert des données dans la base
32    $dao->transferAdminData2Database();
33  } catch (ExceptionImpots $ex) {
34    // on affiche l'erreur
35    print "L'erreur suivante s'est produite : " . utf8_encode($ex->getMessage()) . "\n";
36  }
37  // fin
38  print "Terminé\n";
39  exit;
```

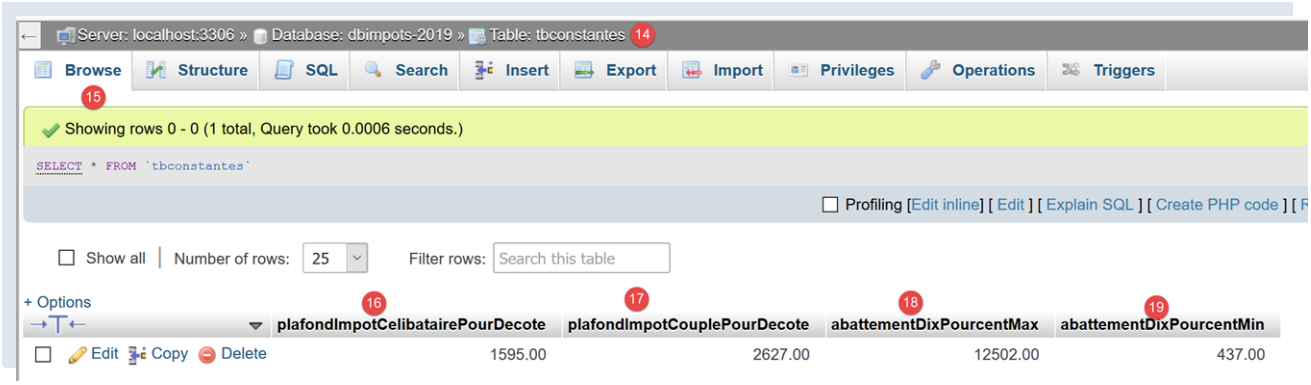
Commentaires

- lignes 12-21 : chargement des classes et interfaces de l'application ;
- lignes 24-24 : les deux fichiers jSON ;
- ligne 30 : on instancie la couche **[dao]** en passant au constructeur les deux fichiers jSON ;
- ligne 32 : on opère le transfert de données ;

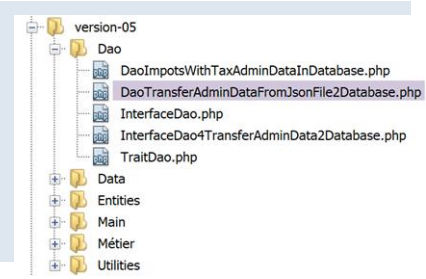
Lorsque nous exécutons ce code, nous obtenons le résultat suivant dans la base de données :



Colonne **[3]**, on voit les valeurs attribuées par MySQL à la clé primaire **[id]**. La numérotation démarre à 1. La copie d'écran ci-dessus a été obtenue après plusieurs exécutions du script.

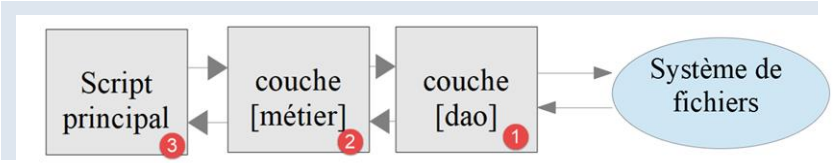


1.13.4 Calcul de l'impôt



1.13.4.1 Architecture

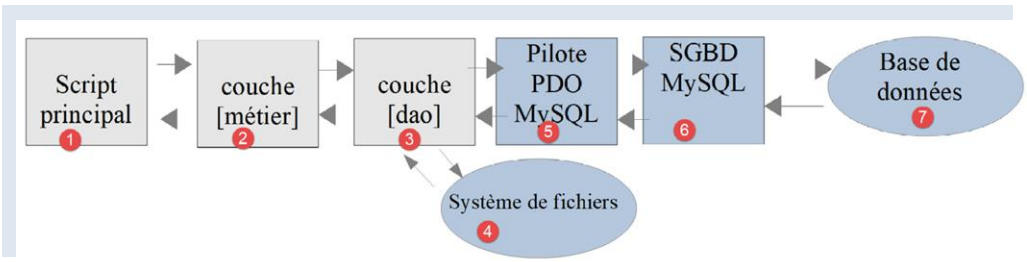
La version 04 de l'application de calcul d'impôt utilisait une architecture en couches :



La couche **[dao]** implémente une interface **[InterfaceDao]**. Nous avons construit une classe implémentant cette interface :

- **[DaoImpotsWithTaxAdminDataInJsonFile]** qui allait chercher les données fiscales dans un fichier jSON. C'était la version 04 ;

Nous allons implémenter l'interface **[InterfaceDao]** par une nouvelle classe **[DaoImpotsWithTaxAdminDataInDatabase]** qui ira chercher les données de l'administration fiscale dans une base de données MySQL. La couche **[dao]**, comme précédemment, écrira les résultats et les erreurs dans des fichiers texte et trouvera les données des contribuables également dans un fichier texte. Seulement cette fois-ci, ces fichiers texte seront des fichiers jSON. Par ailleurs, nous savons que si nous continuons à respecter l'interface **[InterfaceDao]**, la couche **[métier]** n'aura pas à être modifiée.



1.13.4.2 L'entité [TaxPayerData]

La classe **[TaxPayerData]** sert à encapsuler dans une classe les données du fichier jSON **[taxpayersdata.json]** suivant :

```
1  [
```

```
    {
        "marié": "oui",
        "enfants": 2,
        "salaire": 55555
    },
    {
        "marié": "ouix",
        "enfants": "2x",
        "salaire": "55555x"
    },
    {
        "marié": "oui",
        "enfants": "2",
        "salaire": 50000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 50000
    },
    {
        "marié": "non",
        "enfants": 2,
        "salaire": 100000
    },
    {
        "marié": "non",
        "enfants": 3,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 5,
        "salaire": 100000
    },
    {
        "marié": "non",
        "enfants": 0,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 2,
        "salaire": 30000
    },
    {
        "marié": "non",
        "enfants": 0,
        "salaire": 200000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 20000
    }
]
```

La classe **[TaxPayerData]** est la suivante :

```php
<?php

// espace de noms
namespace Application;

// la classe des données
class TaxPayerData extends BaseEntity {
  // données nécessaires au calcul de l'impôt du contribuable
  protected $marié;
  protected $enfants;
  protected $salaire;
  // résultats du calcul de l'impôt
  protected $impôt;
```

```
14    protected $surcôte;
15    protected $décôte;
16    protected $réduction;
17    protected $taux;
18
19    // getters et setters
20    …
21  }
```

Commentaires

- ligne 7 : la classe **[TaxPayerData]** étend la classe **[BaseEntity]**. Les méthodes de sa classe parent étant suffisantes, la classe **[TaxPayerData]** n'en définit pas elle-même. On rappelle que les attributs de la classe **[TaxPayerData]** sont identiques à ceux du fichier jSON **[taxpayersdata.json]** ;

1.13.4.3 La couche [dao]

1.13.4.3.1 Le trait [TraitDao]

Le trait **[TraitDao]** implémente une partie de l'interface **[InterfaceDao]**. Rappelons celle-ci :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   interface InterfaceDao {
7
8     // lecture des données contribuables
9     public function getTaxPayersData(string $taxPayersFilename, string $errorsFilename): array;
10
11    // lecture des données de l'administration fiscale (tranches d'impôts)
12    public function getTaxAdminData(): TaxAdminData;
13
14    // enregistrement des résultats
15    public function saveResults(string $resultsFilename, array $taxPayersData): void;
16  }
17
```

Le trait **[TraitDao]** implémente les méthodes **[getTaxPayersData, saveResults]** de l'interface **[InterfaceDao]**. Du fait qu'entre les versions 04 et 05, on a changé la définition de l'entité **[TaxPayerData]**, il nous faut revoir le code de **[TraitDao]** :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   trait TraitDao {
7
8     // lecture des données contribuables
9     public function getTaxPayersData(string $taxPayersFilename, string $errorsFilename): array {
10      // on récupère les données des contribuables dans un tableau
11      $baseEntity = new BaseEntity();
12      $baseEntity->setFromJsonFile($taxPayersFilename);
13      $arrayOfAttributes = $baseEntity->getArrayOfAttributes();
14      // tableau des données contribuables
15      $taxPayersData = [];
16      // tableau des erreurs
17      $errors = [];
18      // on boucle sur le tableau des attributs d'élements de type [TaxPayerData]
19      $i = 0;
20      foreach ($arrayOfAttributes as $attributesOfTaxPayerData) {
21        // vérification
22        $error = $this->check($attributesOfTaxPayerData);
23        if (!$error) {
24          // un contribuable de +
25          $taxPayersData[] = (new TaxPayerData())->setFrOmArrayOfAttributes($attributesOfTaxPayerData);
26        } else {
27          // une erreur de + - on note le numéro de la donnée invalide
28          $error = ["numéro" => $i] + $error;
29          $errors[] = $error;
30        }
31        // suivant
32        $i++;
33      }
```

```
34      // on sauve les erreurs dans un fichier json
35      $string = "";
36      foreach ($errors as $error) {
37        $string .= \json_encode($error, JSON_UNESCAPED_UNICODE) . "\n";
38      }
39      $this->saveString($errorsFilename, $string);
40      // résultat de la fonction
41      return $taxPayersData;
42    }
43
44    private function check(array $attributesOfTaxPayerData): array {
45      // on vérifie les données de [$taxPayerData]
46      // la liste des atributs erronés
47      $attributes = [];
48      // le statut marital doit être oui ou non
49      $marié = trim(strtolower($attributesOfTaxPayerData["marié"]));
50      $erreur = ($marié !== "oui" and $marié !== "non");
51      if ($erreur) {
52        // on note l'erreur
53        $attributes[] = ["marié" => $marié];
54      }
55      // le nombre d'enfants doit être un entier positif ou nul
56      $enfants = trim($attributesOfTaxPayerData["enfants"]);
57      if (!preg_match("/^\d+$/", $enfants)) {
58        // on note l'erreur
59        $erreur = TRUE;
60        $attributes[] = ["enfants" => $enfants];
61      } else {
62        $enfants = (int) $enfants;
63      }
64
65      // le salaire doit être un entier positif ou nul (sans les centimes d'euros)
66      $salaire = trim($attributesOfTaxPayerData["salaire"]);
67      if (!preg_match("/^\d+$/", $salaire)) {
68        // on note l'erreur
69        $erreur = TRUE;
70        $attributes[] = ["salaire" => $salaire];
71      } else {
72        $salaire = (int) $salaire;
73      }
74
75      // erreur ?
76      if ($erreur) {
77        // retour avec erreur
78        return ["erreurs" => $attributes];
79      } else {
80        // retour sans erreur
81        return [];
82      }
83    }
84
85    // enregistrement des résultats
86    public function saveResults(string $resultsFilename, array $taxPayersData): void {
87      // enregistrement du tableau [$taxPayersData] dans le fichier texte [$resultsFileName]
88      // si le fichier texte [$resultsFileName] n'existe pas, il est créé
89      // construction de la chaîne jSON des résultats
90      $string = "[" . implode(",
91  ", $taxPayersData) . "]";
92      // enregistrement de cette chaîne
93      $this->saveString($resultsFilename, $string);
94    }
95
96    // enregistrement d'es résultats d'un tableau dans un fichier texte
97    private function saveString(string $fileName, string $data): void {
98      // enregistrement de la chaîne [$data] dans le fichier texte [$fileName]
99      // si le fichier texte [$fileName] n'existe pas, il est créé
100     if (file_put_contents($fileName, $data) === FALSE) {
101       throw new ExceptionImpots("Erreur lors de l'enregistrement de données dans le fichier texte
    [$fileName]");
102     }
103   }
104
105 }
```

Commentaires

- **[TraitDao]** implémente les méthodes **[getTaxPayersData]** (ligne 9) et **[saveResults]** (ligne 86) de l'interface **[InterfaceDao]** ;
- ligne 9 : la méthode **[getTaxPayersData]** reçoit en paramètres :
  - **[$taxPayersFilename]** : le nom du fichier jSON des données des contribuables **[taxpayersdata.json]** ;
  - **[$errorsFilename]** : le nom du fichier jSON des erreurs **[errors.json]** ;
- lignes 11-13 : le contenu du fichier jSON des données des contribuables est transféré dans un tableau associatif **[$arrayOfAttributes]**. Si le fichier jSON s'avère inexploitable, une exception **[ExceptionImpots]** a été lancée ;
- ligne 15 : le tableau **[$taxPayersData]** va contenir les données des contribuables encapsulées dans des objets de type **[TaxPayerData]** ;
- ligne 17 : on va cumuler les erreurs dans le tableau **[$errors]** ;
- lignes 99-33 : construction du tableau **[$taxPayersData]** ;
- ligne 22 : avant d'être encapsulées dans un type **[TaxPayerData]**, les données sont vérifiées. La méthode **[check]** rend :
  - un tableau **[‘erreurs’=>[…]]** avec les attributs erronés si les données sont incorrectes ;
  - un tableau vide si les données sont correctes ;
- ligne 25 : cas où les données sont valides. Un nouvel objet **[TaxPayerData]** est construit et ajouté au tableau **[$taxPayersData]** ;
- lignes 26-30 : cas où les données sont invalides. On note dans l'erreur, le n° de l'objet **[TaxPayerData]** erroné dans le fichier jSON pour que l'utilisateur puisse le retrouver, puis l'erreur est ajoutée au tableau **[$errors]** ;
- lignes 35-39 : on enregistre les erreurs rencontrées dans le fichier jSON **[$errorsFilename]** passé en paramètre, ligne 9 ;
- ligne 41 : on rend le tableau des objets **[TaxPayerData]** construits : c'était l'objectif de la méthode ;
- lignes 44-83 : la méthode privée **[check]** vérifie la validité des paramètres **[marié, enfants, salaire]** du tableau **[$attributesOfTaxPayerData]** passé en paramètre ligne 44. S'il y a des attributs erronés, elle les cumule dans le tableau **[$attributes]** (lignes 47, 53, 60, 70) sous la forme d'un tableau **[‘attribut erroné’=> valeur de l’attribut erroné]** ;
- ligne 78 : s'il y a des erreurs, on rend un tableau **[‘erreurs’=>$attributes]** ;
- ligne 81 : s'il n'y a pas d'erreurs, on rend un tableau d'erreurs vide ;
- lignes 86-93 : implémentation de la méthode **[saveResults]** de l'interface **[InterfaceDao]** ;
- ligne 90 : on construit la chaîne jSON à enregistrer dans le fichier jSON **[$resultsFilename]** passé en paramètre ligne 86. on doit construire la chaîne jSON d'un tableau :
  - chaque élément du tableau est séparé du suivant par une virgule et un saut de ligne ;
  - l'ensemble du tableau est entouré de crochets **[]** ;
- ligne 92 : la chaîne jSON est enregistrée dans le fichier jSON **[$resultsFilename]** ;

1.13.4.3.2 La classe [DaoImpotsWithTaxAdminDataInDatabase]

La classe **[DaoImpotsWithTaxAdminDataInDatabase]** implémente l'interface **[InterfaceDao]** de la façon suivante :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   // définition d'une classe ImpotsWithDataInDatabase
7   class DaoImpotsWithTaxAdminDataInDatabase implements InterfaceDao {
8     // usage d'un trait
9     use TraitDao;
10    // l'objet de type TaxAdminData qui contient les données des tranches d'impôts
11    private $taxAdminData;
12    // l'objet de type [Database] contennat les caractéristiques de la BD
13    private $database;
14
15    // constructeur
16    public function __construct(string $databaseFilename) {
17      // on mémorise la configuration jSON de la bd
18      $this->database = (new Database())->setFromJsonFile($databaseFilename);
19      // on prépare l'attribut
20      $this->taxAdminData = new TaxAdminData();
21      try {
22        // on ouvre la connexion à la base de données
23        $connexion = new \PDO(
24          $this->database->getDsn(),
25          $this->database->getId(),
26          $this->database->getPwd());
27        // on veut qu'à chaque erreur de SGBD, une exception soit lancée
28        $connexion->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
29        // on démarre une transaction
30        $connexion->beginTransaction();
31        // on remplit la table des tranches d'impôt
32        $this->getTranches($connexion);
33        // on remplit la table des constantes
34        $this->getConstantes($connexion);
35        // on termine la transaction sur un succès
36        $connexion->commit();
37      } catch (\PDOException $ex) {
```

```
38        // y-a-t-il une transaction en cours ?
39        if (isset($connexion) && $connexion->inTransaction()) {
40          // on termine la transaction sur un échec
41          $connexion->rollBack();
42        }
43        // on remonte l'exception au code appelant
44        throw new ExceptionImpots($ex->getMessage());
45      } finally {
46        // on ferme la connexion
47        $connexion = NULL;
48      }
49    }
50
51    // lecture des données de la base
52    private function getTranches($connexion): void {
53      …
54    }
55
56    // lecture de la table des constantes
57    private function getConstantes($connexion): void {
58      …
59    }
60
61    // retourne les données permettant le calcul de l'impôt
62    public function getTaxAdminData(): TaxAdminData {
63      return $this->taxAdminData;
64    }
65
66  }
67
```

Commentaires

- ligne 4 : on garde l'espace de noms déjà utilisé pour les autres implémentations de la couche **[dao]** ;
- ligne 7 : la classe **[DaoImpotsWithTaxAdminDataInDatabase]** implémente l'interface **[InterfaceDao]** ;
- ligne 9 : on importe le trait **[TraitDao]**. On sait que ce trait implémente une partie de l'interface. La seule méthode qui reste à implémenter est la méthode **[getTaxAdminData]** des lignes 62-64. Cette méthode se contente de rendre l'attribut privé **[taxAdminData]** de la ligne 11. On en déduit que le constructeur devra initialiser cet attribut. C'est son unique rôle ;
- ligne 16 : le constructeur reçoit comme unique paramètre **[$databaseFilename]** qui est le nom du fichier jSON **[database.json]** qui définit la base de données MySQL **[dbimpots-2019]** ;
- ligne 18 : le fichier jSON **[$databaseFilename]** est utilisé pour créer un objet de type **[Database]** construit et mémorisé dans l'attribut **[$database]** de la ligne 13. Si le fichier jSON n'a pu être exploité correctement, une exception **[ExceptionImpots]** a été lancée ;
- ligne 20 : on crée l'objet **[$this→taxAdminData]** que le constructeur doit initialiser ;
- lignes 22-26 : on ouvre la connexion à la base de données. Notez la notation **[\PDO]** pour désigner la classe **[PDO]** de PHP. En effet, comme on est dans l'espace de noms **[Application]**, si on écrivait simplement **[PDO]**, ce nom relatif serait préfixé par l'espace de noms courant et donnerait donc la classe **[Application\PDO]** qui n'existe pas ;
- ligne 28 : lors d'une erreur, le SGBD lancera une \PDOException (ligne 37) ;
- ligne 30 : on démarre une transaction. Celle-ci n'est pas vraiment utile car seuls deux ordres SQL vont être exécutés, ordres qui ne modifient pas la base. On le fait néanmoins pour s'isoler des autres utilisateurs de la base ;
- ligne 32 : la lecture de la table des tranches d'imposition **[tbtranches]** est faite par la méthode privée **[getTranches]** de la ligne 52 ;
- ligne 34 : la lecture de la table des constantes de calcul **[tbconstantes]** est faite par la méthode privée **[getConstantes]** de la ligne 57 ;
- ligne 36 : si on arrive à cette ligne c'est que tout s'est bien passé. On valide donc la transaction;
- lignes 37-42 : si on arrive là c'est qu'une exception s'est produite. On invalide donc la transaction s'il y en avait une en cours (lignes 39-42). Ligne 44, pour avoir des exceptions homogènes, on relance le message de l'exception reçue sous la forme cette fois d'une exception de type **[ExceptionImpots]** ;
- lignes 45-48 : dans tous les cas (exception ou pas) on ferme la connexion ;

La méthode **[getTranches]** est la suivante :

```
1   private function getTranches($connexion): void {
2       // raccourcis
3       $database = $this->database;
4       $taxAdminData = $this->taxAdminData;
5       // on prépare la requête SELECT
6       $statement = $connexion->prepare(
7         "select {$database->getColLimites()}," .
8         " {$database->getColCoeffR()}," .
9         " {$database->getColCoeffN()}" .
```

```
10        " from {$database->getTableTranches()}");
11      // on exécute l'ordre préparé avec les valeurs des tranches d'impôts
12      $statement->execute();
13      // on exploite le résultat
14      $limites = [];
15      $coeffR = [];
16      $coeffN = [];
17      // remplissage des trois tableaux
18      while ($tranche = $statement->fetch(\PDO::FETCH_OBJ)) {
19        $limites[] = (float) $tranche->{$database->getColLimites()};
20        $coeffR[] = (float) $tranche->{$database->getColCoeffR()};
21        $coeffN[] = (float) $tranche->{$database->getColCoeffN()};
22      }
23      // on mémorise les données dans l'attribut [$taxAdminData] de la classe
24      $taxAdminData->setFromArrayOfAttributes([
25        "limites" => $limites,
26        "coeffR" => $coeffR,
27        "coeffN" => $coeffN
28      ]);
29    }
```

Commentaires

- ligne 1 : la méthode reçoit en paramètre **[$connexion]** qui est une connexion ouverte et dans laquelle une transaction est en cours ;
- lignes 2-4 : on crée deux raccourcis pour éviter d'avoir à écrire **[$this->database]** et **[$taxAdminData = $this->taxAdminData]** dans tout le code. On a là des copies de références d'objets et non pas une copie des objets eux-mêmes ;
- ligne 6-10 : l'ordre SELECT est préparé, puis exécuté en ligne 12 ;
- lignes 13-22 : le résultat du SELECT est exploité. Les informations reçues sont cumulées dans trois tableaux **[limites, coeffR, coeffN]** ;
- lignes 24-28 : les trois tableaux sont utilisés pour initialiser l'attribut **[$this->taxAdminData]** de la classe ;

La méthode privée **[getConstantes]** est la suivante :

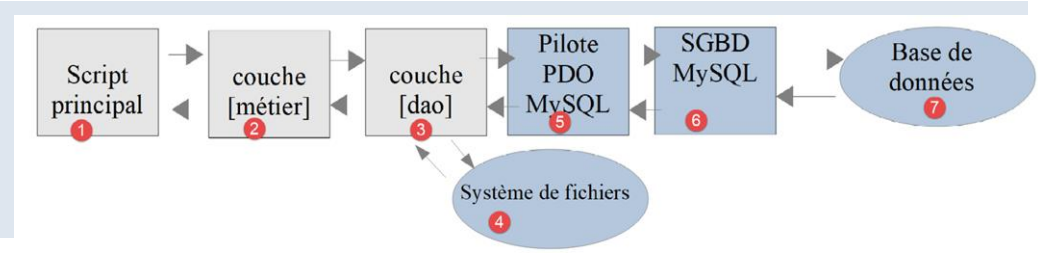
```
1   private function getConstantes($connexion): void {
2       // raccourcis
3       $database = $this->database;
4       $taxAdminData = $this->taxAdminData;
5       // on prépare la requête SELECT
6       $select = "select {$database->getColPlafondQfDemiPart()}," .
7         "{$database->getColPlafondRevenusCelibatairePourReduction()}," .
8         "{$database->getColPlafondRevenusCouplePourReduction()}," . "{$database->getColValeurReducDemiPart()}," .
9         "{$database->getColPlafondDecoteCelibataire()}," . "{$database->getColPlafondDecoteCouple()}," .
10        "{$database->getColPlafondImpotCelibatairePourDecote()}," . "{$database->getColPlafondImpotCouplePourDecote()}," .
11        "{$database->getColAbattementDixPourcentMax()}," . "{$database->getColAbattementDixPourcentMin()}" .
12        " from {$database->getTableConstantes()}";
13      $statement = $connexion->prepare($select);
14      // on exécute l'ordre préparé
15      $statement->execute();
16      // on exploite le résultat - 1 seule ligne ici
17      $row = $statement->fetch(\PDO::FETCH_OBJ);
18      // on initialise l'attribut [$taxAdminData]
19      $taxAdminData->setPlafondQfDemiPart($row->{$database->getColPlafondQfDemiPart()});
20      $taxAdminData->setPlafondRevenusCelibatairePourReduction(
21        $row->{$database->getColPlafondRevenusCelibatairePourReduction()});
22      $taxAdminData->setPlafondRevenusCouplePourReduction($row->{$database->getColPlafondRevenusCouplePourReduction()});
23      $taxAdminData->setValeurReducDemiPart($row->{$database->getColValeurReducDemiPart()});
24      $taxAdminData->setPlafondDecoteCelibataire($row->{$database->getColPlafondDecoteCelibataire()});
25      $taxAdminData->setPlafondDecoteCouple($row->{$database->getColPlafondDecoteCouple()});
26      $taxAdminData->setPlafondImpotCelibatairePourDecote($row->{$database->getColPlafondImpotCelibatairePourDecote()});
27      $taxAdminData->setPlafondImpotCouplePourDecote($row->{$database->getColPlafondImpotCouplePourDecote()});
28      $taxAdminData->setAbattementDixPourcentMax($row->{$database->getColAbattementDixPourcentMax()});
29      $taxAdminData->setAbattementDixPourcentMin($row->{$database->getColAbattementDixPourcentMin()});
30    }
```

Commentaires

- ligne 1 : la méthode reçoit en paramètre **[$connexion]** qui est une connexion ouverte et dans laquelle une transaction est en cours ;
- lignes 2-4 : on crée deux raccourcis pour éviter d'avoir à écrire **[$this->database]** et **[$taxAdminData = $this->taxAdminData]** dans tout le code. On a là des copies de références d'objets et non pas une copie des objets eux-mêmes ;
- ligne 6-15 : l'ordre SELECT est préparé, puis exécuté en ligne 15 ;
- lignes 17-29 : le résultat du SELECT est exploité. Les informations récupérées sont utilisées pour initialiser l'attribut **[$this->taxAdminData]** de la classe ;

Note : on remarquera que la classe ne dépend pas du SGBD MySQL. C'est le code appelant qui fixe le SGBD utilisé via le DSN de la base de données.

1.13.4.4 La couche [métier]



- nous venons d'implémenter la couche **[dao]** (3) ;
- parce que nous avons respecté l'interface **[InterfaceDao]**, la couche **[métier]** (2) peut en théorie rester inchangée. Cependant, nous n'avons pas seulement modifié la couche **[dao]**. Nous avons également modifié les entités qui elles sont partagées par toutes les couches ;

La couche **[métier]** implémente l'interface **[InterfaceMetier]** suivante :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   interface InterfaceMetier {
7
8     // calcul des impôts d'un contribuable
9     public function calculerImpot(string $marié, int $enfants, int $salaire): array;
10
11    // calcul des impôts en mode batch
12    public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string
    $errorsFileName): void;
13  }
```

- ligne 12 : la méthode **[executeBatchImpots]** utilise désormais le fichier jSON **[$taxPayersFileName]** alors que dans la version 04, c'était un fichier texte basique. ;

Dans la version 04, la méthode **[executeBatchImpots]** était la suivante :

```
1   public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string
    $errorsFileName): void {
2       // on laisse remonter les exceptions qui proviennent de la couche [dao]
3       // on récupère les données contribuables
4       $taxPayersData = $this->dao->getTaxPayersData($taxPayersFileName, $errorsFileName);
5       // tableau des résultats
6       $results = [];
7       // on les exploite
8       foreach ($taxPayersData as $taxPayerData) {
9         // on calcule l'impôt
10        $result = $this->calculerImpot(
11          $taxPayerData->getMarié(),
12          $taxPayerData->getEnfants(),
13          $taxPayerData->getSalaire());
14        // on complète [$taxPayerData]
15        $taxPayerData->setMontant($result["impôt"]);
16        $taxPayerData->setDécôte($result["décôte"]);
17        $taxPayerData->setSurCôte($result["surcôte"]);
18        $taxPayerData->setTaux($result["taux"]);
19        $taxPayerData->setRéduction($result["réduction"]);
20        // on met le résultat dans le tableau des résultats
21        $results [] = $taxPayerData;
22      }
23      // enregistrement des résultats
24      $this->dao->saveResults($resultsFileName, $results);
25    }
```

- la ligne 15 est désormais erronée. Dans la nouvelle définition de la classe **[TaxPayerData]**, la méthode **[setMontant]** n'existe plus ;

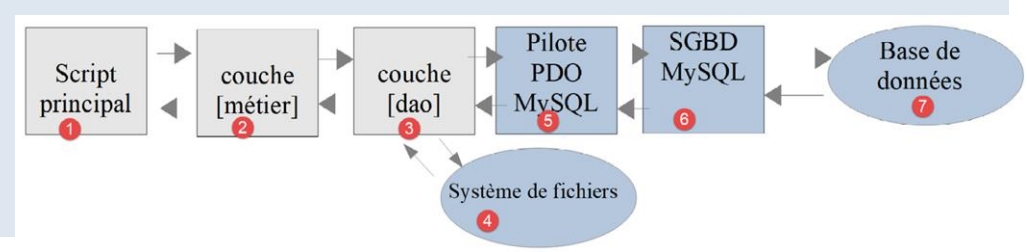
Dans la version 05, la méthode **[executeBatchImpots]** sera la suivante :

```
1   public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string
    $errorsFileName): void {
2       // on laisse remonter les exceptions qui proviennent de la couche [dao]
3       // on récupère les données contribuables
4       $taxPayersData = $this->dao->getTaxPayersData($taxPayersFileName, $errorsFileName);
5       // tableau des résultats
6       $results = [];
7       // on les exploite
8       foreach ($taxPayersData as $taxPayerData) {
9         // on calcule l'impôt
10        $result = $this->calculerImpot(
11          $taxPayerData->getMarié(),
12          $taxPayerData->getEnfants(),
13          $taxPayerData->getSalaire());
14        // on complète [$taxPayerData]
15        $taxPayerData->setFromArrayOfAttributes($result);
16        // on met le résultat dans le tableau des résultats
17        $results [] = $taxPayerData;
18      }
19      // enregistrement des résultats
20      $this->dao->saveResults($resultsFileName, $results);
21    }
```

Commentaires

- ligne 15 : au lieu d'utiliser les setters individuels de la la classe **[TaxPayerData]**, on utilise son setter global **[setFromArrayOfAttributes]** ;
- le reste du code n'a pas à être modifié ;

1.13.4.5 Le script principal



- nous venons d'implémenter les couches **[dao]** (3) et **[métier]** (2) ;
- il nous reste à écrire le script principal (1) ;

Le script principal est analogue à celui de la version 04 :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de fonctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // gestion des erreurs par PHP
10  //ini_set("display_errors", "0");
11  // inclusion interface et classes
12  require_once __DIR__ . "/../Entities/BaseEntity.php";
13  require_once __DIR__ . "/../Entities/TaxAdminData.php";
14  require_once __DIR__ . "/../Entities/TaxPayerData.php";
15  require_once __DIR__ . "/../Entities/Database.php";
16  require_once __DIR__ . "/../Entities/ExceptionImpots.php";
17  require_once __DIR__ . "/../Utilities/Utilitaires.php";
18  require_once __DIR__ . "/../Dao/InterfaceDao.php";
19  require_once __DIR__ . "/../Dao/TraitDao.php";
20  require_once __DIR__ . "/../Dao/DaoImpotsWithTaxAdminDataInDatabase.php";
21  require_once __DIR__ . "/../Métier/InterfaceMetier.php";
22  require_once __DIR__ . "/../Métier/Metier.php";
```

```
23  //
24  // définition des constantes
25  const DATABASE_CONFIG_FILENAME = "../Data/database.json";
26  const TAXADMINDATA_FILENAME = "../Data/taxadmindata.json";
27  const RESULTS_FILENAME = "../Data/resultats.json";
28  const ERRORS_FILENAME = "../Data/errors.json";
29  const TAXPAYERSDATA_FILENAME = "../Data/taxpayersdata.json";
30
31  try {
32    // création de la couche [dao]
33    $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
34    // création de la couche [métier]
35    $métier = new Metier($dao);
36    // calcul de l'impôts en mode batch
37    $métier->executeBatchImpots(TAXPAYERSDATA_FILENAME, RESULTS_FILENAME, ERRORS_FILENAME);
38  } catch (ExceptionImpots $ex) {
39    // on affiche l'erreur
40    print "Une erreur s'est produite : " . utf8_encode($ex->getMessage()) . "\n";
41  }
42  // fin
43  print "Terminé\n";
44  exit;
45
46
```

Commentaires

- lignes 12-22 : chargement de tous les fichiers de la version 05;
- lignes 25-29 : les noms des différents fichiers jSON de l'application ;
- ligne 33 : construction de la couche **[dao]** ;
- ligne 35 : construction de la couche **[métier]** ;
- ligne 37 : appel de la méthode **[executeBatchImpots]** de la couche **[métier]** ;

Résultats

L'application produit deux fichiers jSON :

- **[resultats.json]** : les résultats des différents calcul d'impôts ;
- **[errors.json]** : qui signale les erreurs trouvées dans le fichier jSON **[taxpayersdata.json]** ;

Le fichier **[errors.json]** est le suivant :

```
1   {
2           "numéro": 1,
3           "erreurs": [
4                   {
5                           "marié": "ouix"
6                   },
7                   {
8                           "enfants": "2x"
9                   },
10                  {
11                          "salaire": "55555x"
12                  }
13          ]
14  }
```

Cela signifie que dans **[taxpayersdata.json]**, l'élément n° 1 du tableau des contribuables est erroné. Le fichier **[taxpayersdata.json]** était le suivant :

```
1   [
2       {
3           "marié": "oui",
4           "enfants": 2,
5           "salaire": 55555
6       },
7       {
8           "marié": "ouix",
9           "enfants": "2x",
10          "salaire": "55555x"
11      },
12      {
13          "marié": "oui",
```

```
        "enfants": "2",
        "salaire": 50000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 50000
    },
    {
        "marié": "non",
        "enfants": 2,
        "salaire": 100000
    },
    {
        "marié": "non",
        "enfants": 3,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 5,
        "salaire": 100000
    },
    {
        "marié": "non",
        "enfants": 0,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 2,
        "salaire": 30000
    },
    {
        "marié": "non",
        "enfants": 0,
        "salaire": 200000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 20000
    }
]
```

Le fichier des résultats **[resultats.json]** est lui le suivant :

```
[
        {
                "marié": "oui",
                "enfants": 2,
                "salaire": 55555,
                "impôt": 2814,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.14
        },
        {
                "marié": "oui",
                "enfants": "2",
                "salaire": 50000,
                "impôt": 1384,
                "surcôte": 0,
                "décôte": 384,
                "réduction": 347,
                "taux": 0.14
        },
        {
                "marié": "oui",
                "enfants": 3,
                "salaire": 50000,
```

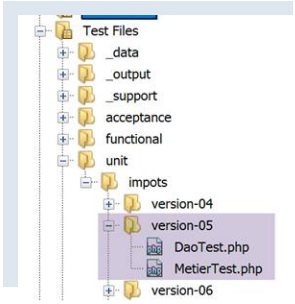
```
                "impôt": 0,
                "surcôte": 0,
                "décôte": 720,
                "réduction": 0,
                "taux": 0.14
        },
        {
                "marié": "non",
                "enfants": 2,
                "salaire": 100000,
                "impôt": 19884,
                "surcôte": 4480,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.41
        },
        {
                "marié": "non",
                "enfants": 3,
                "salaire": 100000,
                "impôt": 16782,
                "surcôte": 7176,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.41
        },
        {
                "marié": "oui",
                "enfants": 3,
                "salaire": 100000,
                "impôt": 9200,
                "surcôte": 2180,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.3
        },
        {
                "marié": "oui",
                "enfants": 5,
                "salaire": 100000,
                "impôt": 4230,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.14
        },
        {
                "marié": "non",
                "enfants": 0,
                "salaire": 100000,
                "impôt": 22986,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.41
        },
        {
                "marié": "oui",
                "enfants": 2,
                "salaire": 30000,
                "impôt": 0,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0
        },
        {
                "marié": "non",
                "enfants": 0,
                "salaire": 200000,
                "impôt": 64210,
                "surcôte": 7498,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.45
        },
        {
```

```
                "marié": "oui",
                "enfants": 3,
                "salaire": 20000,
                "impôt": 0,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0
        }
]
```

Ces résultats sont conformes à ceux de la version 04.

1.13.5 Tests [Codeception]

Comme il a été fait au paragraphe lien pour la version 04, nous allons écrire des tests **[Codeception]** pour la version 05.



1.13.5.1 Test de la couche [dao]

Le test **[DaoTest.php]** est le suivant :

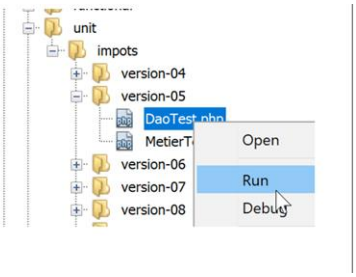
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // répertoires racines
10  define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-05");
11  define("VENDOR", "C:/myprograms/laragon-lite/www/vendor");
12
13  // inclusion interface et classes
14  require_once ROOT . "/Entities/BaseEntity.php";
15  require_once ROOT . "/Entities/TaxAdminData.php";
16  require_once ROOT . "/Entities/TaxPayerData.php";
17  require_once ROOT . "/Entities/Database.php";
18  require_once ROOT . "/Entities/ExceptionImpots.php";
19  require_once ROOT . "/Utilities/Utilitaires.php";
20  require_once ROOT . "/Dao/InterfaceDao.php";
21  require_once ROOT . "/Dao/TraitDao.php";
22  require_once ROOT . "/Dao/DaoImpotsWithTaxAdminDataInDatabase.php";
23  require_once ROOT . "/Métier/InterfaceMetier.php";
24  require_once ROOT . "/Métier/Metier.php";
25  // bibliothèques tierces
26  require_once VENDOR . "/autoload.php";
27
28  // définition des constantes
29  const DATABASE_CONFIG_FILENAME = ROOT ."/Data/database.json";
30  const TAXADMINDATA_FILENAME = ROOT ."/Data/taxadmindata.json";
31  const RESULTS_FILENAME = ROOT ."/Data/resultats.json";
32  const ERRORS_FILENAME = ROOT ."/Data/errors.json";
33  const TAXPAYERSDATA_FILENAME = ROOT ."/Data/taxpayersdata.json";
34
35  class DaoTest extends \Codeception\Test\Unit {
36    // TaxAdminData
37    private $taxAdminData;
38
39    public function __construct() {
40      parent::__construct();
```

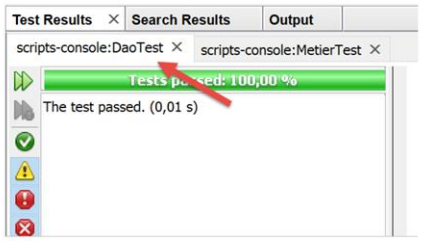
```
41      // création de la couche [dao]
42      $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
43      $this->taxAdminData = $dao->getTaxAdminData();
44    }
45
46    // tests
47    public function testTaxAdminData() {
48      // constantes de calcul
49      $this->assertEquals(1551, $this->taxAdminData->getPlafondQfDemiPart());
50      …
51    }
52
53  }
```

Commentaires

- lignes 9-33 : définition de l'environnement du test. Nous utilisons le même que celui utilisé par le script principal **[MainCalculateImpotsWithTaxAdminDataInMySQLDatabase]** décrit au paragraphe lien ;
- lignes 39-44 : construction de la couche **[dao]** ;
- ligne 43 : l'attribut **[$this→taxAdminData]** contient les données à tester ;
- lignes 47-51 : la méthode **[testTaxAdminData]** est celle décrite au paragraphe lien ;

Les résultats du test sont les suivants :





1.13.5.2 Test de la couche [métier]

Le test **[MetierTest.php]** est le suivant :

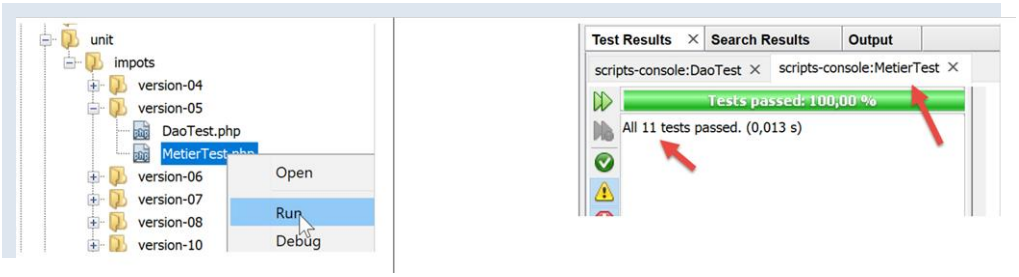
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // répertoires racines
10  define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-05");
11  define("VENDOR", "C:/myprograms/laragon-lite/www/vendor");
12
13  // inclusion interface et classes
14  require_once ROOT . "/Entities/BaseEntity.php";
15  require_once ROOT . "/Entities/TaxAdminData.php";
16  require_once ROOT . "/Entities/TaxPayerData.php";
17  require_once ROOT . "/Entities/Database.php";
18  require_once ROOT . "/Entities/ExceptionImpots.php";
19  require_once ROOT . "/Utilities/Utilitaires.php";
20  require_once ROOT . "/Dao/InterfaceDao.php";
21  require_once ROOT . "/Dao/TraitDao.php";
22  require_once ROOT . "/Dao/DaoImpotsWithTaxAdminDataInDatabase.php";
23  require_once ROOT . "/Métier/InterfaceMetier.php";
24  require_once ROOT . "/Métier/Metier.php";
25  // bibliothèques tierces
26  require_once VENDOR . "/autoload.php";
27
28  // définition des constantes
29  const DATABASE_CONFIG_FILENAME = ROOT ."/Data/database.json";
30  const TAXADMINDATA_FILENAME = ROOT ."/Data/taxadmindata.json";
31  const RESULTS_FILENAME = ROOT ."/Data/resultats.json";
32  const ERRORS_FILENAME = ROOT ."/Data/errors.json";
33  const TAXPAYERSDATA_FILENAME = ROOT ."/Data/taxpayersdata.json";
34
```

```
35  class MetierTest extends \Codeception\Test\Unit {
36    // couche métier
37    private $métier;
38
39    public function __construct() {
40      parent::__construct();
41      // création de la couche [dao]
42      $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
43      // création de la couche [métier]
44      $this->métier = new Metier($dao);
45    }
46
47    // tests
48    public function test1() {
49      $result = $this->métier->calculerImpot("oui", 2, 55555);
50      $this->assertEqualsWithDelta(2815, $result["impôt"], 1);
51      $this->assertEqualsWithDelta(0, $result["surcôte"], 1);
52      $this->assertEqualsWithDelta(0, $result["décôte"], 1);
53      $this->assertEqualsWithDelta(0, $result["réduction"], 1);
54      $this->assertEquals(0.14, $result["taux"]);
55    }
56  ……..
57  public function test11() {
58      $result = $this->métier->calculerImpot("oui", 3, 200000);
59      $this->assertEqualsWithDelta(42842, $result["impôt"], 1);
60      $this->assertEqualsWithDelta(17283, $result["surcôte"], 1);
61      $this->assertEqualsWithDelta(0, $result["décôte"], 1);
62      $this->assertEqualsWithDelta(0, $result["réduction"], 1);
63      $this->assertEquals(0.41, $result["taux"]);
64    }
65
66  }
```

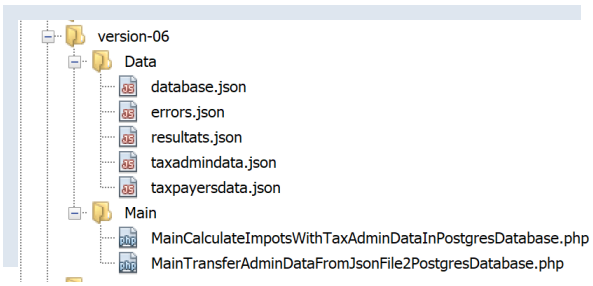
Commentaires

- lignes 9-33 : définition de l'environnement du test. Nous utilisons le même que celui utilisé par le script principal **[MainCalculateImpotsWithTaxAdminDataInMySQLDatabase]** décrit au paragraphe lien ;
- lignes 39-45 : construction des couches **[dao]** et **[métier]** ;
- ligne 44 : l'attribut **[$this→métier]** référence la couche **[métier]** ;
- lignes 47-64 : les méthodes **[test1, test2…, test11]** sont celles décrites au paragraphe lien ;

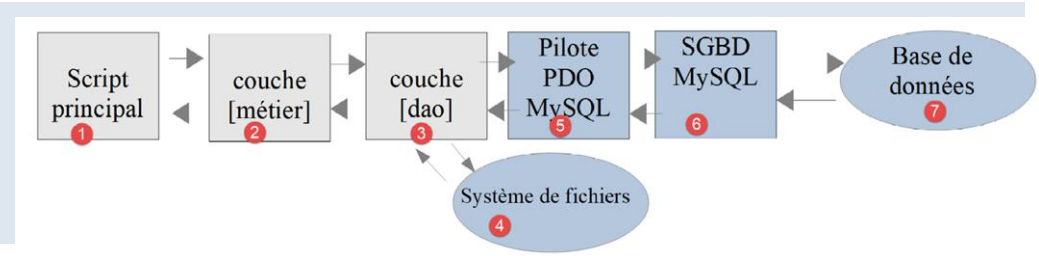
Les résultats du test sont les suivants :



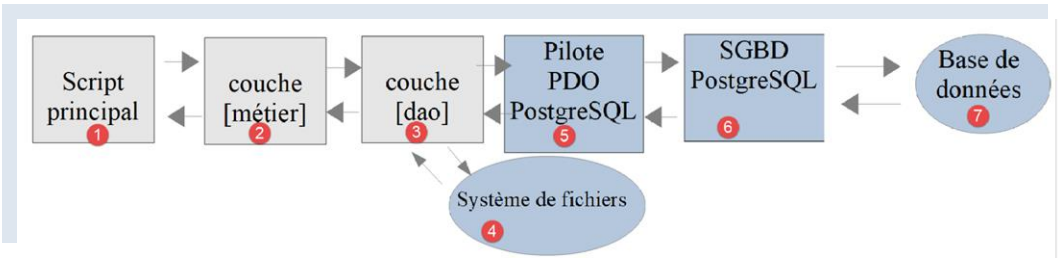
1.14 Exercice d'application – version 6



Nous venons d'implémenter la structure en couches suivante :

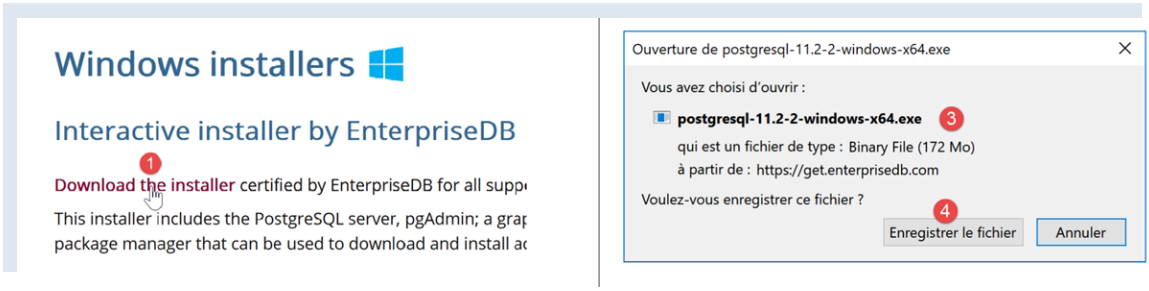


Le SGBD utilisé dans les exemples était MySQL. Au paragraphe lien nous avions remarqué que rien dans la classe implémentant la couche **[dao]** ne laissait supposer qu'on utilisait un SGBD particulier. C'est ce que nous allons vérifier maintenant en utilisant un autre SGBD, le SGBD PostgreSQL. L'architecture en couches devient la suivante :



1.14.1 Installation du SGBD PostgreSQL

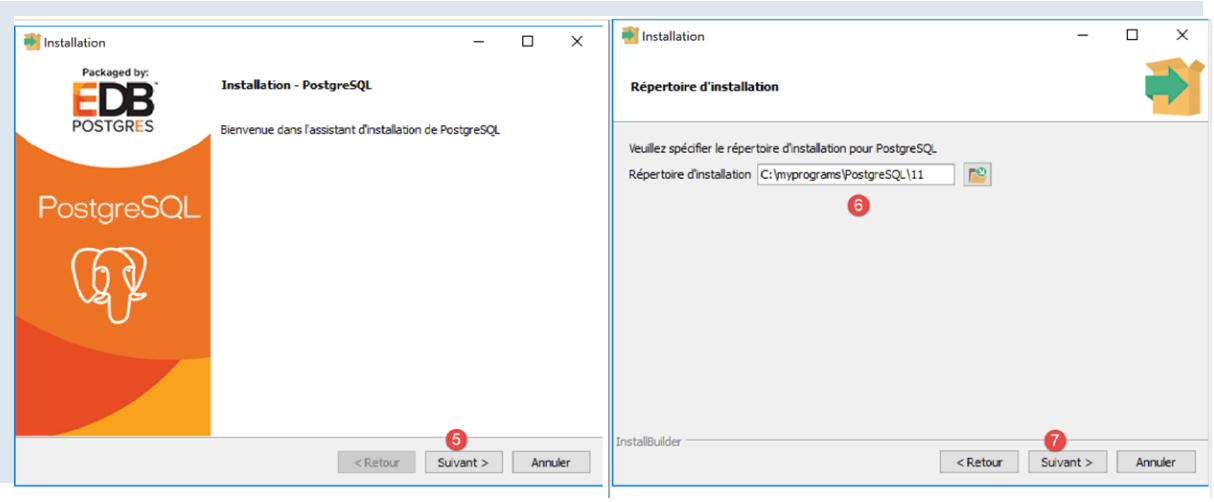
Les distributions du SGBD PostgreSQL sont disponibles à l'URL **[https://www.postgresql.org/download/]** (mai 2019). Nous montrons l'installation de la version pour Windows 64 bits :



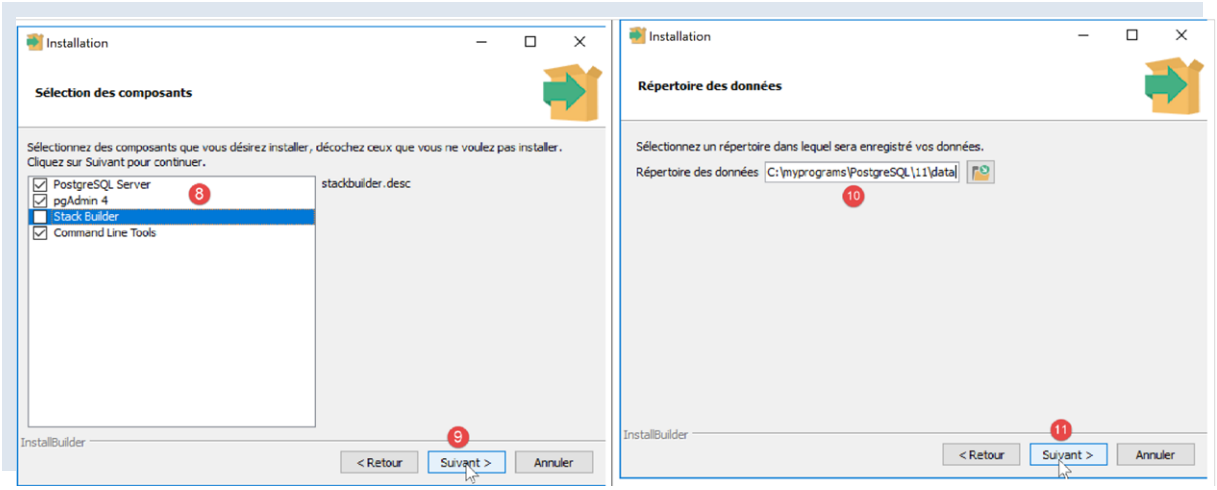
PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
11.2	N/A	N/A	Download	2 Download	N/A

- en **[1-4]**, on télécharge l'installateur du SGBD ;

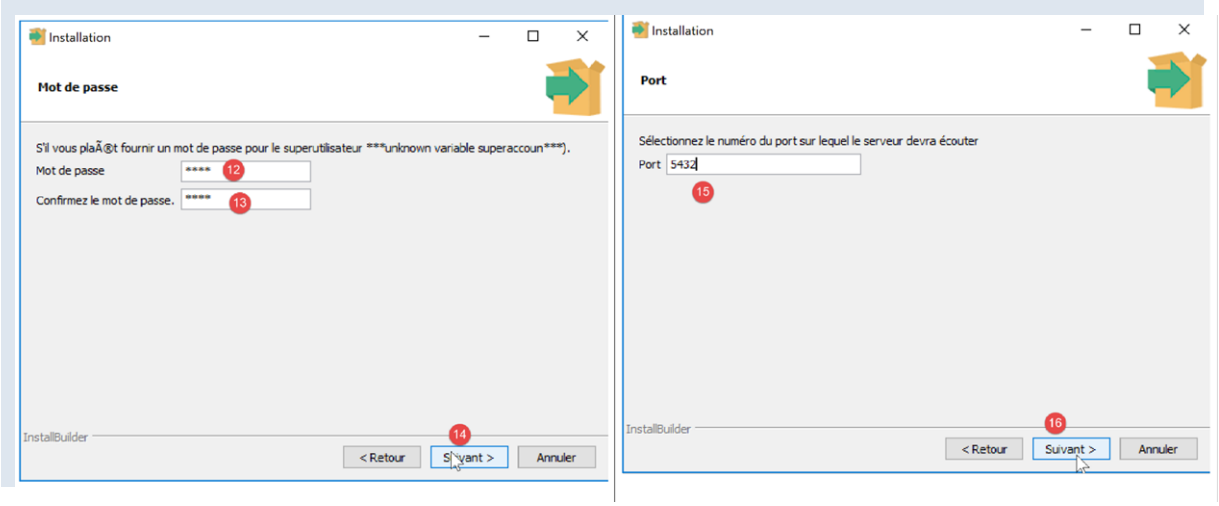
On lance l'installateur téléchargé :



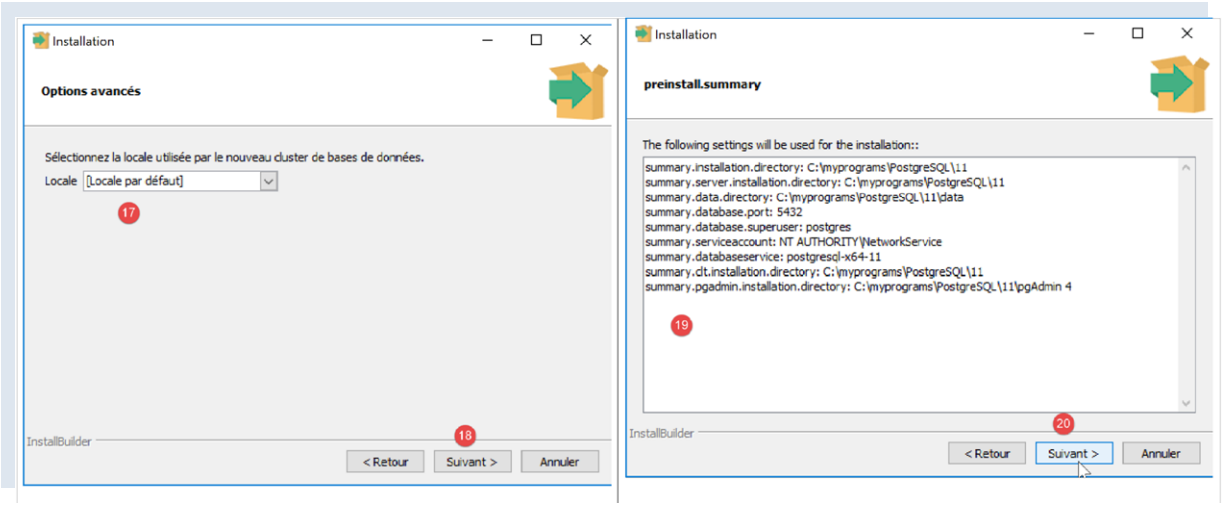
- en **[6]**, indiquez un dossier d'installation ;



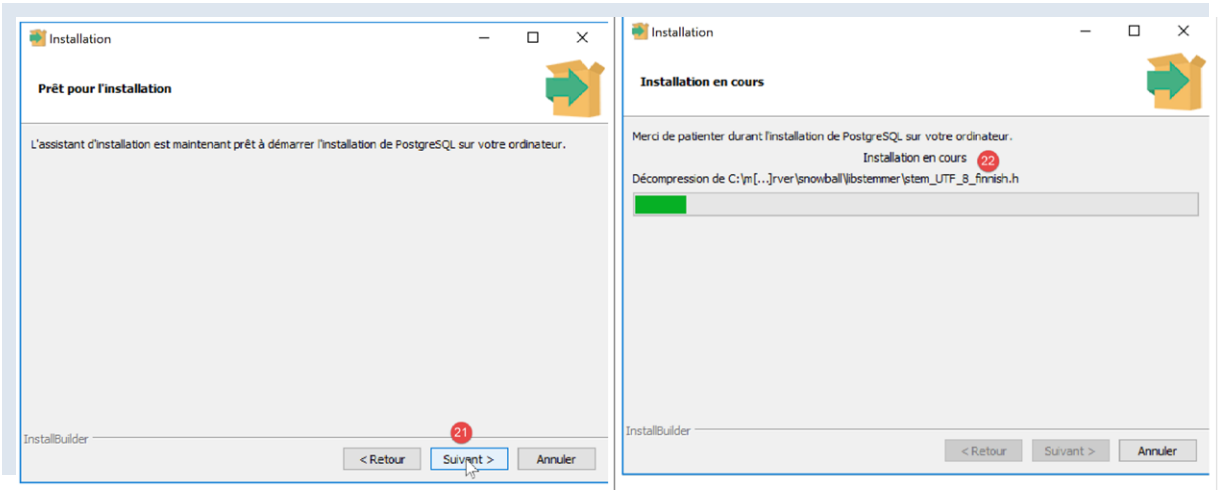
- en **[8]**, l'option **[Stack Builder]** est inutile pour ce qu'on veut faire ici ;
- en **[10]**, laissez la valeur qui vous sera présentée ;

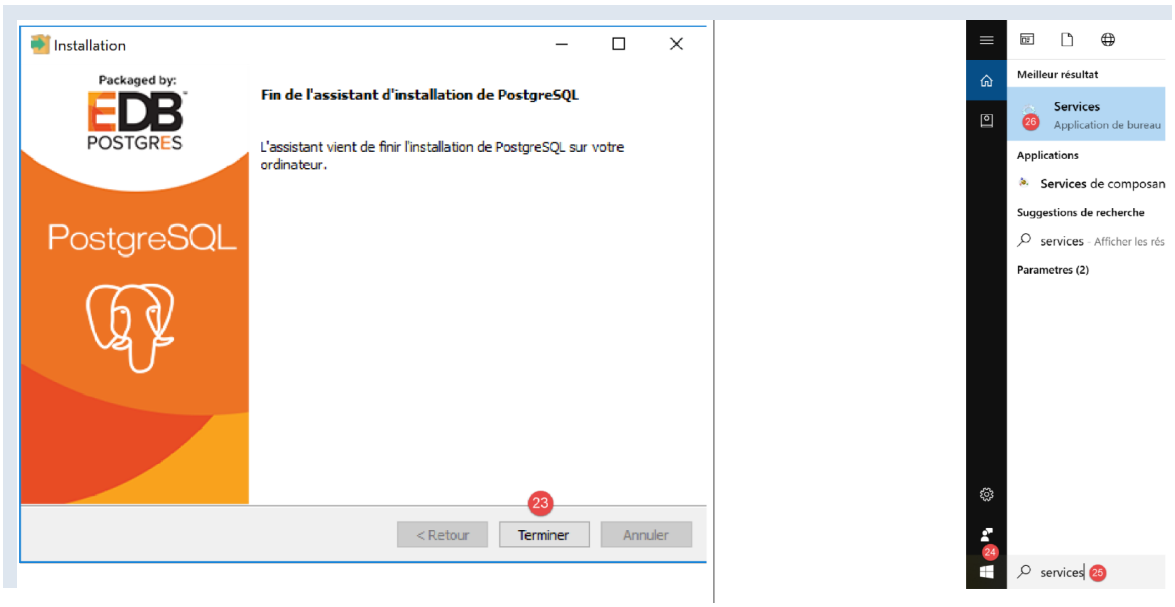


- en **[12-13]**, on a mis ici le mot de passe **[root]**. Ce sera le mot de passe de l'administrateur du SGBD qui s'appelle **[postgres]**. PostgreSQL l'appelle également le super-utilisateur ;
- en **[15]**, laissez la valeur par défaut : c'est le port d'écoute du SGBD ;

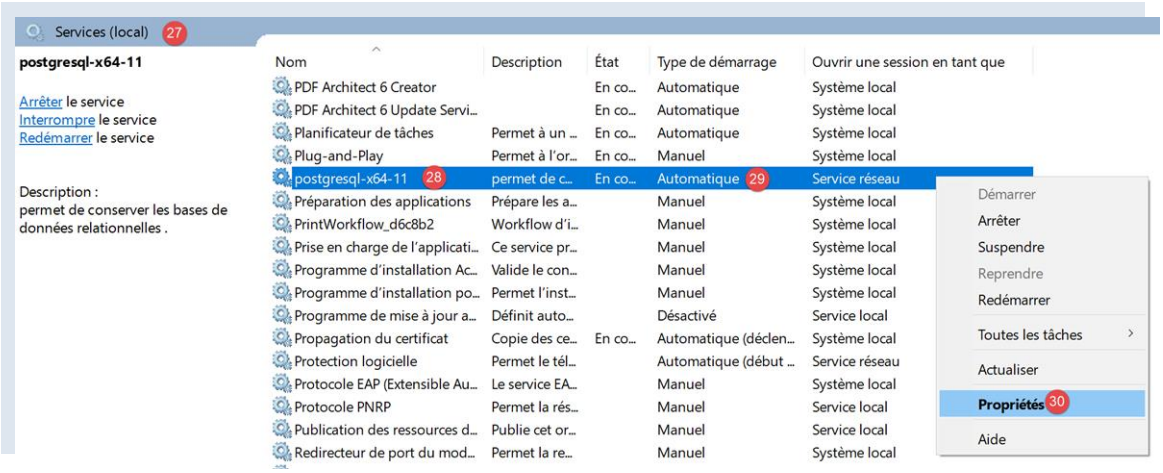


- en **[17]**, laissez la valeur par défaut ;
- en **[19]**, le résumé de la configuration de l'installation ;

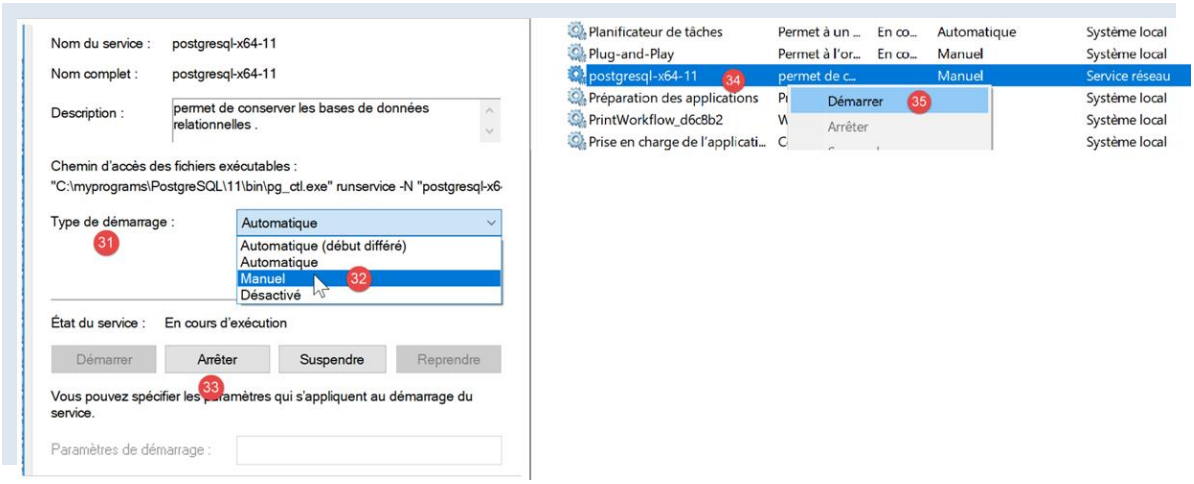




Sous windows, le SGBD PostgreSQL est installé comme un service windows lancé automatiquement. La plupart du temps ce n'est pas souhaitable. Nous allons modifier cette configuration. Tapez **[services]** dans la barre de recherche de Windows **[24-26]** :



- en **[29]**, on voit que le service du SGBD PostgreSQL est en mode automatique. On change cela en accédant aux propriétés du service **[30]** :

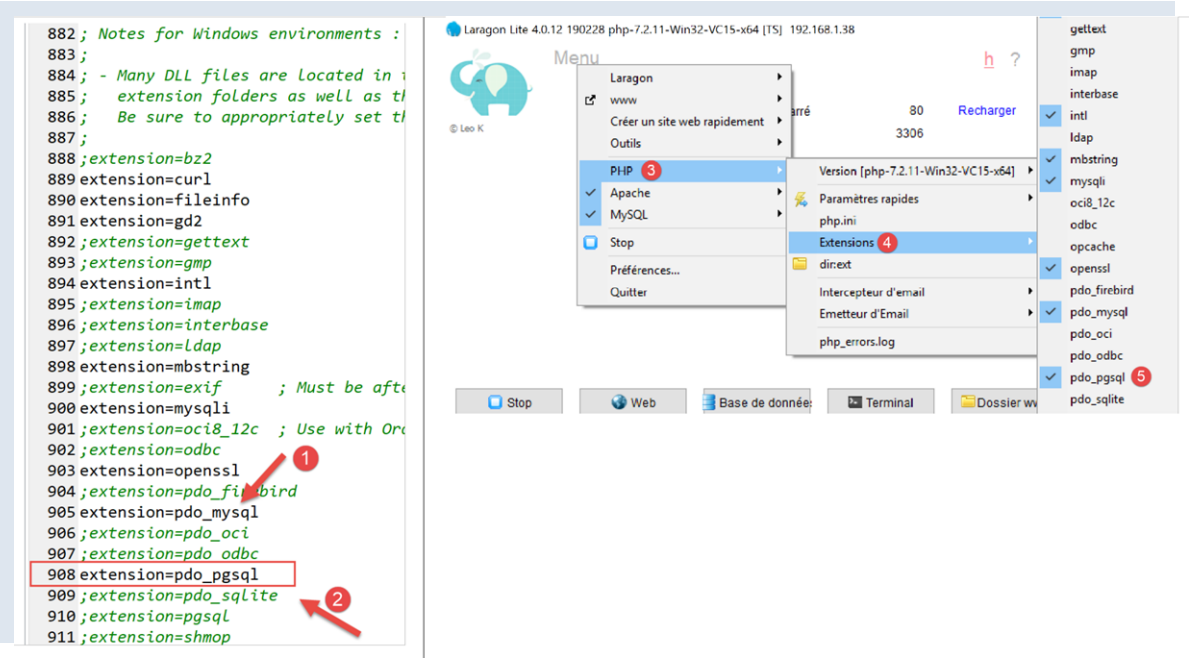


- en **[31-32]**, mettez le démarrage en mode manuel ;
- en **[33]**, arrêtez le service ;

Lorsque vous voudrez démarrer manuellement le SGBD, revenez à l'application **[services]**, cliquez droit sur le service **[postgresql]** (34) et lancez le (35).

1.14.2 Activation de l'extension PDO du SGBD PostgreSQL

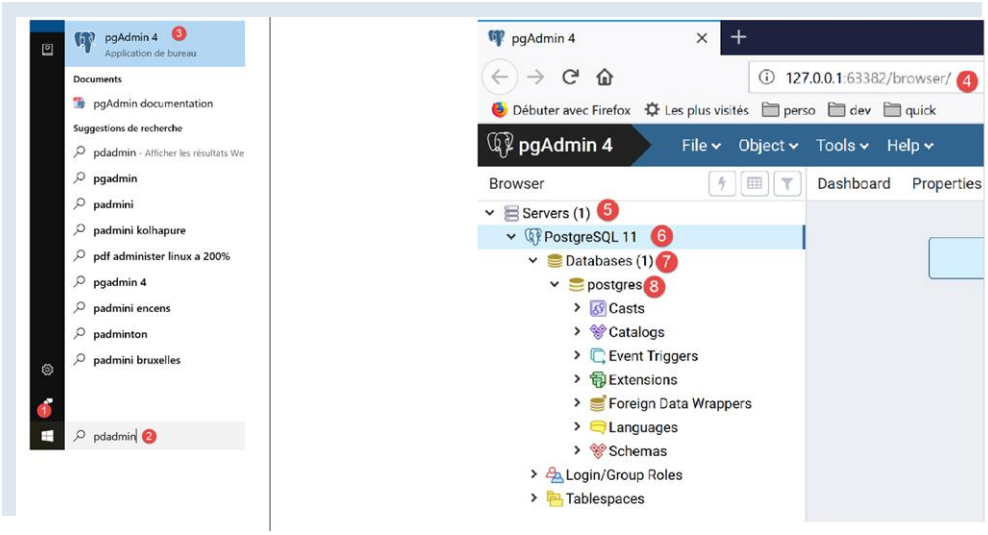
Nous allons modifier le fichier **[php.ini]** qui configure PHP (cf paragraphe lien) :



- en **[2]**, vérifiez que l'extension PDO de PostgreSQL est activée. Ceci fait, sauvegardez la modification puis relancez Laragon pour être sûrs que la modification va être prise en compte. Ensuite vérifiez la configuration de PHP directement à partir de Laragon **[3-5]**.

1.14.3 Administrer PostgreSQL avec l'outil [pgAdmin]

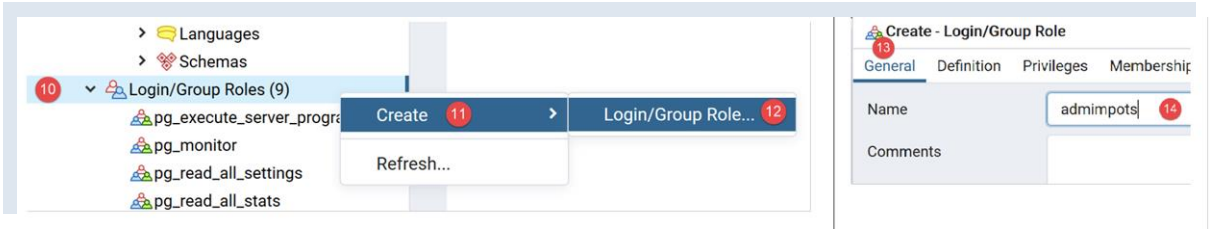
Lancez le service windows du SGBD PostgreSQL (cf paragraphe lien). Puis de la même façon que vous avez lancé l'outil **[services]**, lancez l'outil **[pgadmin]** qui permet d'administrer le SGBD PostgreSQL **[1-3]** :

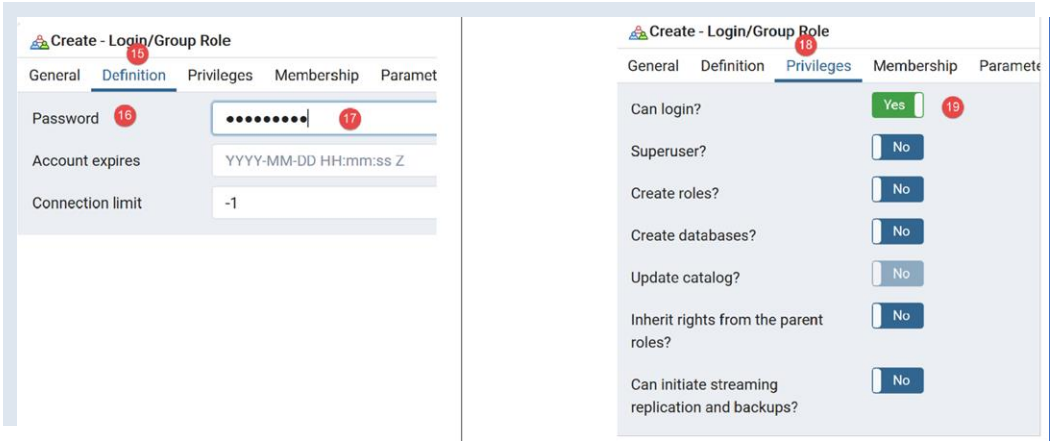


Il est possible qu'à un moment donné on vous demande le mot de passe du super-utilisateur. Celui-ci s'appelle **[postgres]**. Vous avez défini son mot de passe lors de l'installation du SGBD. Dans ce document, nous avons donné le mot de passe **[root]** au super-utilisateur lors de l'installation.

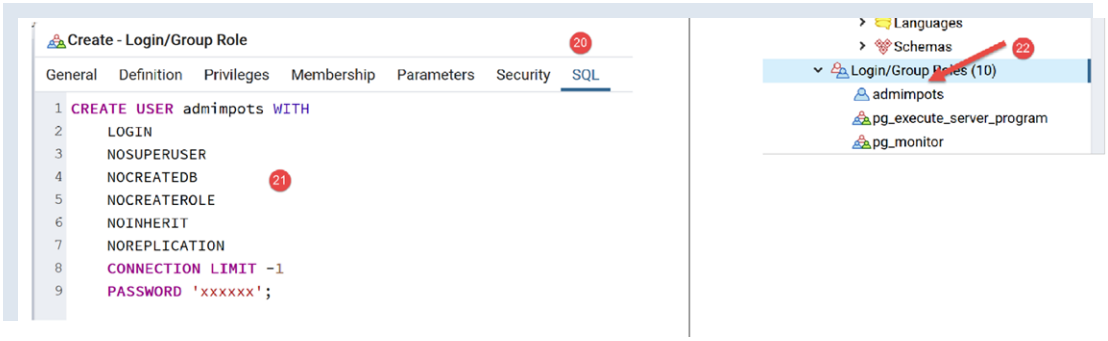
- en **[4]**, **[pgAdmin]** est une application web ;
- en **[5]**, la liste des serveurs PostgreSQL détectés par **[pgAdmin]**, ici 1 ;
- en **[6]**, le serveur PostgreSQL que nous avons lancé ;
- en **[7]**, les bases de données du SGBD, ici 1 ;
- en **[8]**, la base **[postgresql]** est gérée par le super-utilisateur **[postgres]** ;

Créons tout d'abord un utilisateur **[admimpots]** avec le mot de passe **[mdpimpots]** :



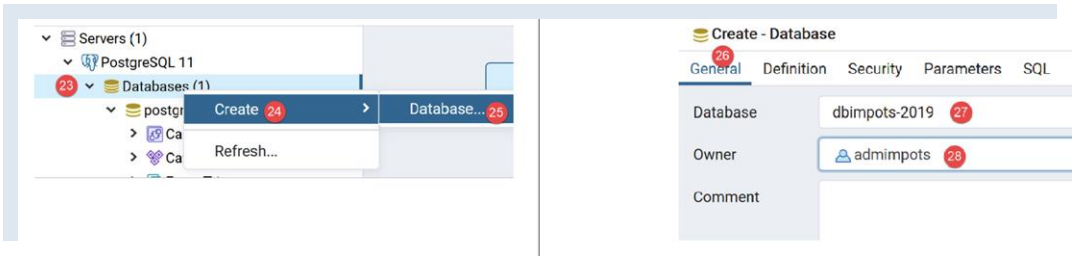


- en **[17]**, on a mis **[mdpimpots]** ;

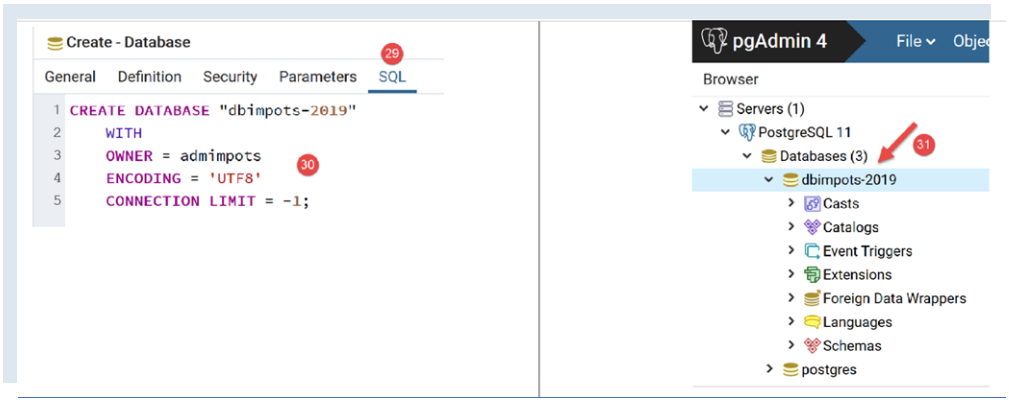


- en **[21]**, le code SQL que va émettre l'outil **[pgAdmin]** vers le SGBD PostgreSQL. C'est une façon d'apprendre le langage SQL propriétaire de PostgreSQL ;
- en **[22]**, après validation de l'assistant **[Save]**, l'utilisateur **[admimpots]** a été créé ;

Maintenant nous créons la base **[dbimpots-2019]** :



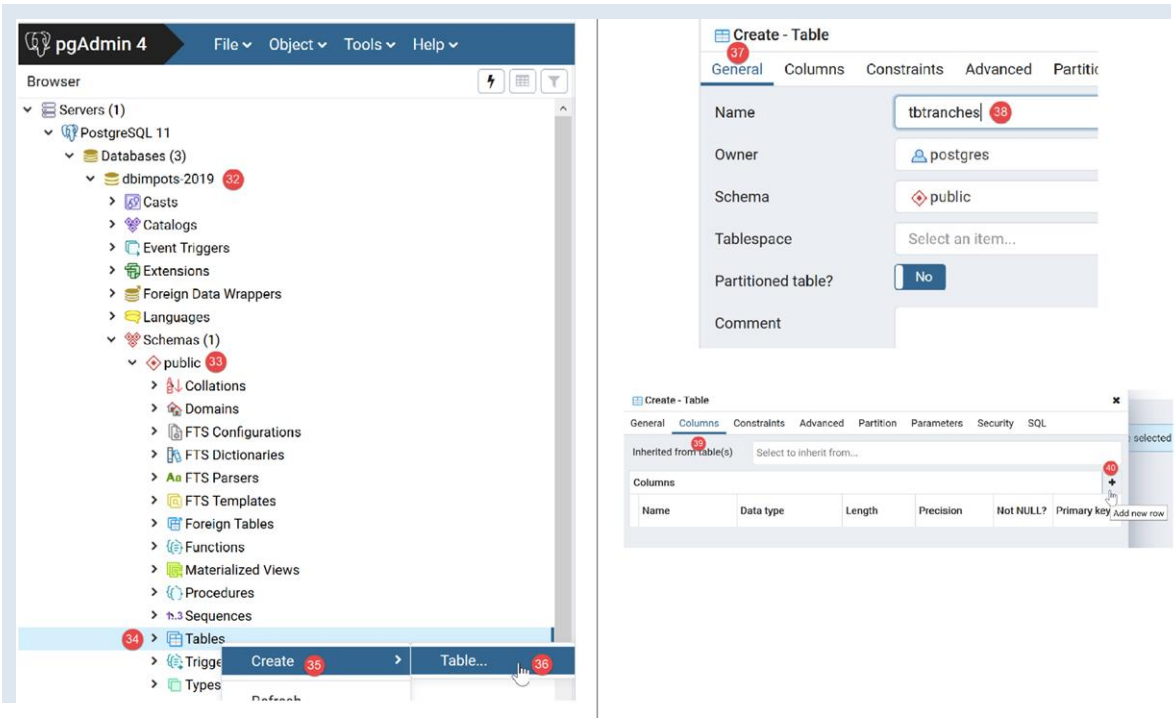
On clique droit sur **[23]**, puis **[24-25]** pour créer une nouvelle base de données. Dans l'onglet **[26]**, on définit le nom de la base **[27]** et son propriétaire **[admimpots] [28]**.



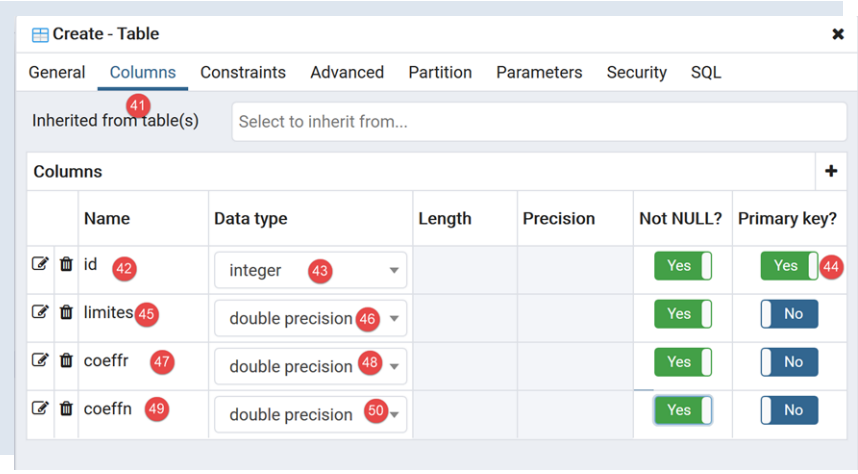
- en **[30]**, le code SQL de création de la base ;
- en **[31]**, après validation de l'assistant **[Save]**, la base **[dbimpots-2019]** est créée ;

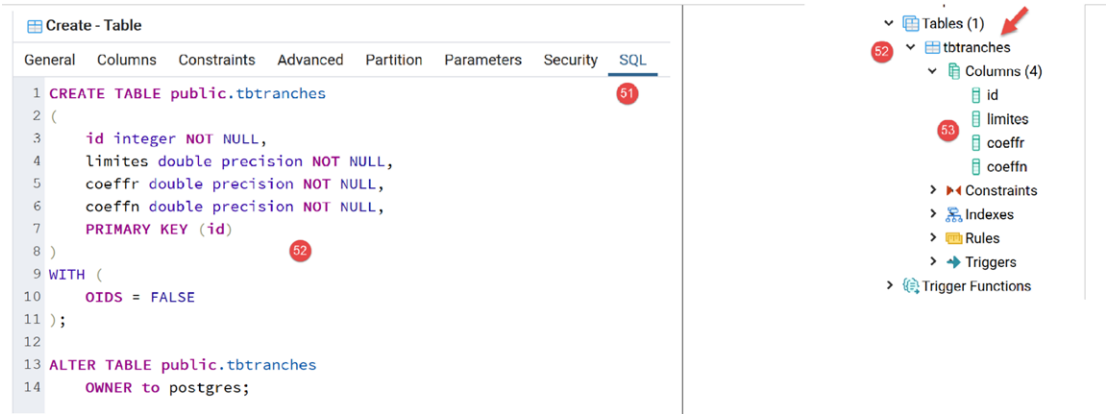
Maintenant, nous allons créer la table **[tbtranches]** avec les colonnes **[id, limites, coeffr, coeffn]**. Une particularité de PostgreSQL est que les noms de colonnes sont sensibles à la casse (majuscules / minuscules) ce qui n'est habituellement pas le cas avec les autres SGBD. Ainsi avec MySQL, l'ordre **[select limites, coeffR, coeffN from tbtranches]** fonctionnera même si les colonnes réelles de la table **[tbtranches]** sont **[LIMITES, COEFFR, COEFFN]**. Avec PostgreSQL, l'ordre SQL ne fonctionnera pas. On pourrait alors écrire **[select LIMITES, COEFFR, COEFFN from tbtranches]** mais ça ne fonctionnera toujours pas, car PostgreSQL va exécuter l'ordre **[select limites, coeffr, coeffn from tbtranches]** : il passe par défaut les noms des colonnes en minuscules. Pour qu'il ne fasse pas cela, il faut écrire : **[select "LIMITES", "COEFFR", "COEFFN" from tbtranches]**, ç-à-d qu'il faut protéger les noms des colonnes avec des guillemets. Pour ces raisons, nous allons donner aux colonnes des noms en minuscules. Les noms des objets d'une base de données peuvent être une source d'incompatibilité entre SGBD, certains noms étant des mots réservés dans certains SGBD et pas dans d'autres.

Nous créons la table **[tbtranches]** :



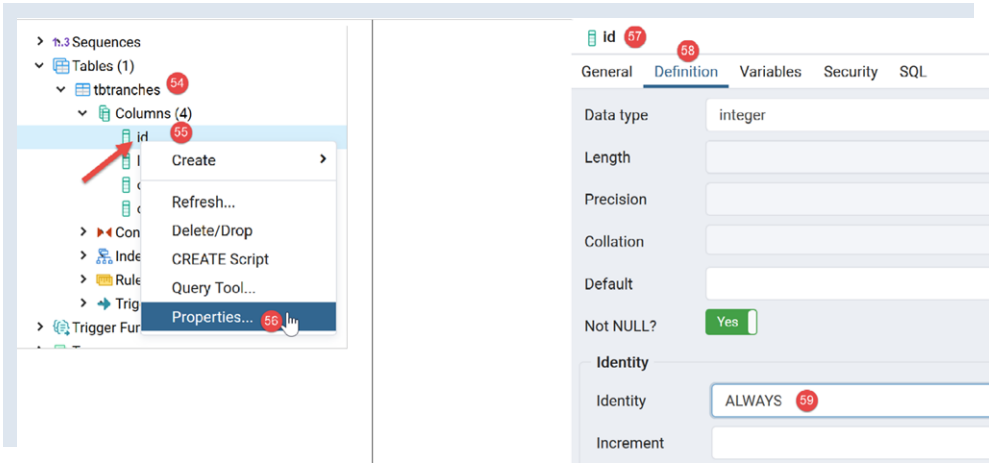
- utilisez le bouton **[40]** pour créer des colonnes ;



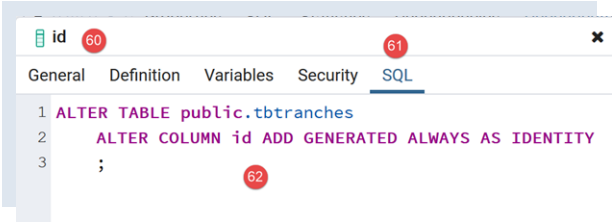


- après avoir terminé l'assistant de création par **[Save]**, la table **[tbtranches]** est créée **[52-53]** ;

Il nous faut indiquer au SGBD qu'il doit lui-même générer la clé primaire **[id]** lors de l'insertion d'une ligne dans la table :



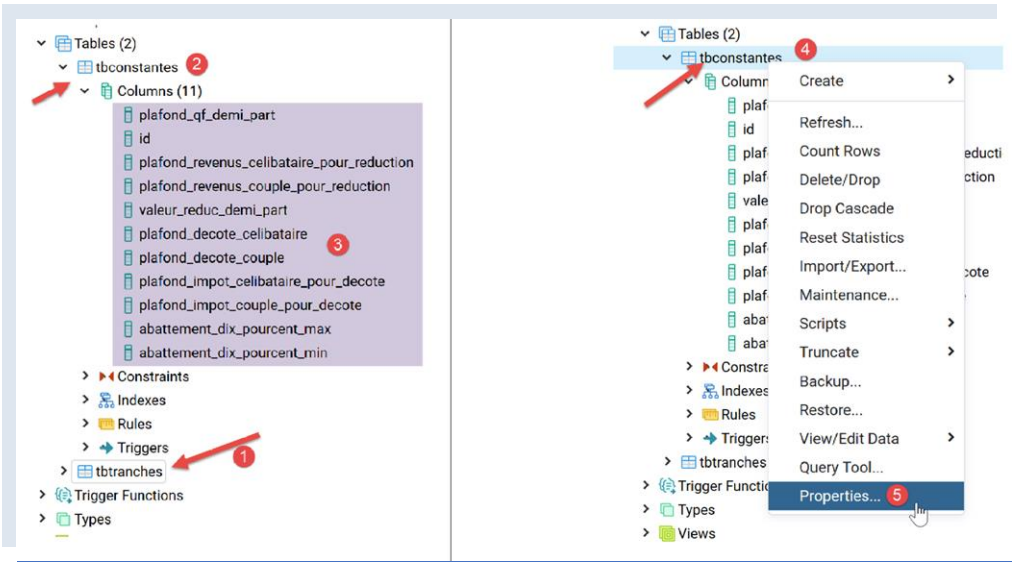
- en **[56]** on accède aux propriétés de la clé primaire **[id]** ;
- en **[59]**, on indique que la colonne est de type **[Identity]**. Cela va entraîner que le SGBD va générer les valeurs de la clé primaire ;

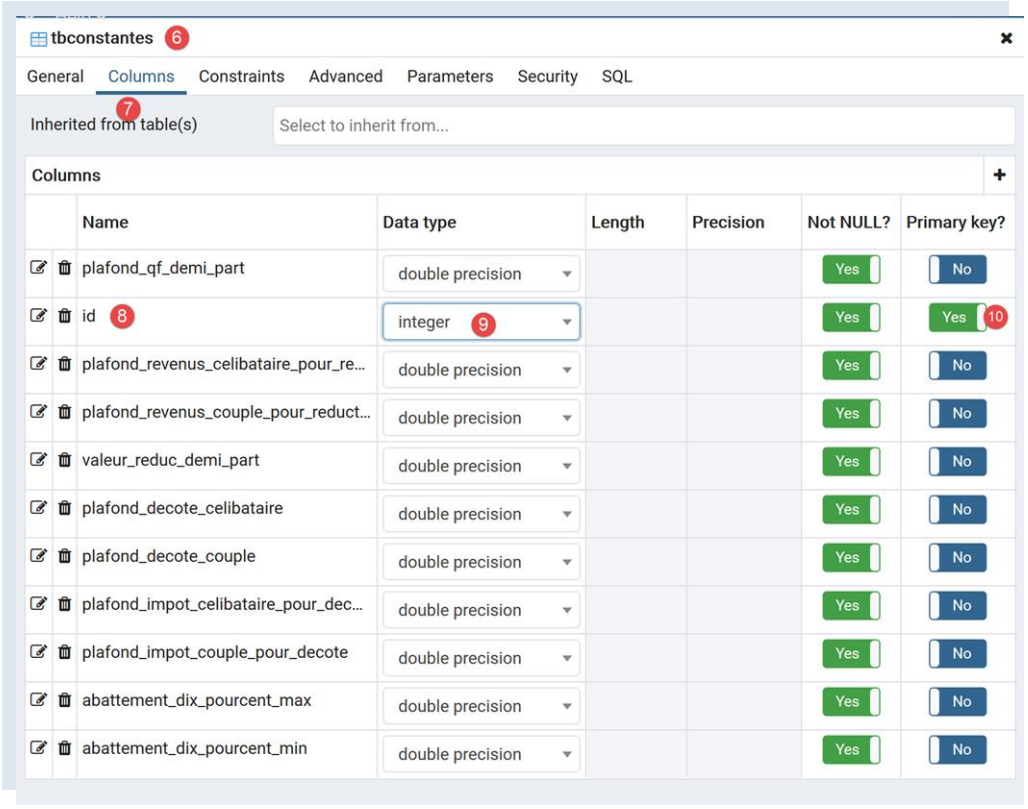


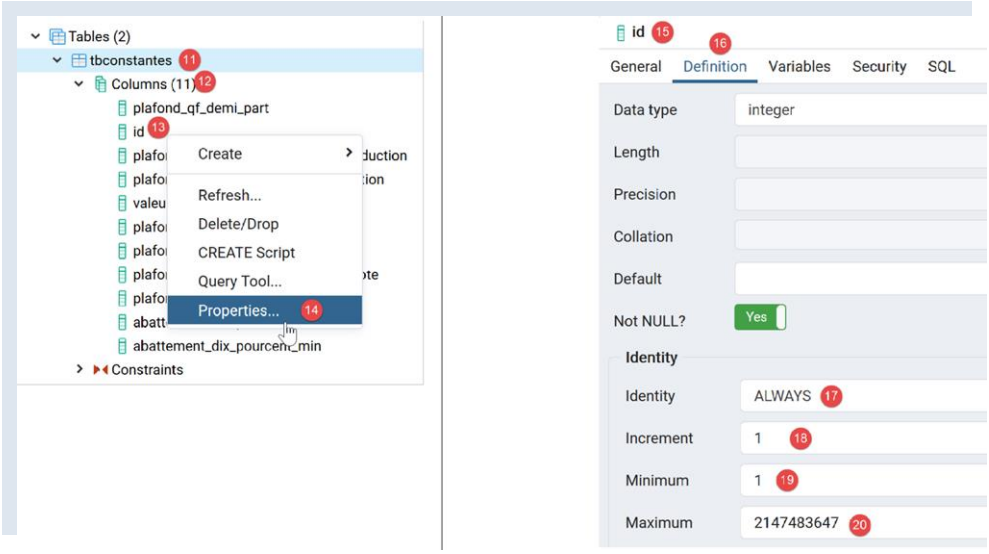
- en **[62]**, le code SQL généré pour cette opération ;

La table **[tbtranches]** est désormais prête.

Nous refaisons les mêmes opérations pour créer la table **[tbconstantes]**. Nous donnons le résultat à obtenir :

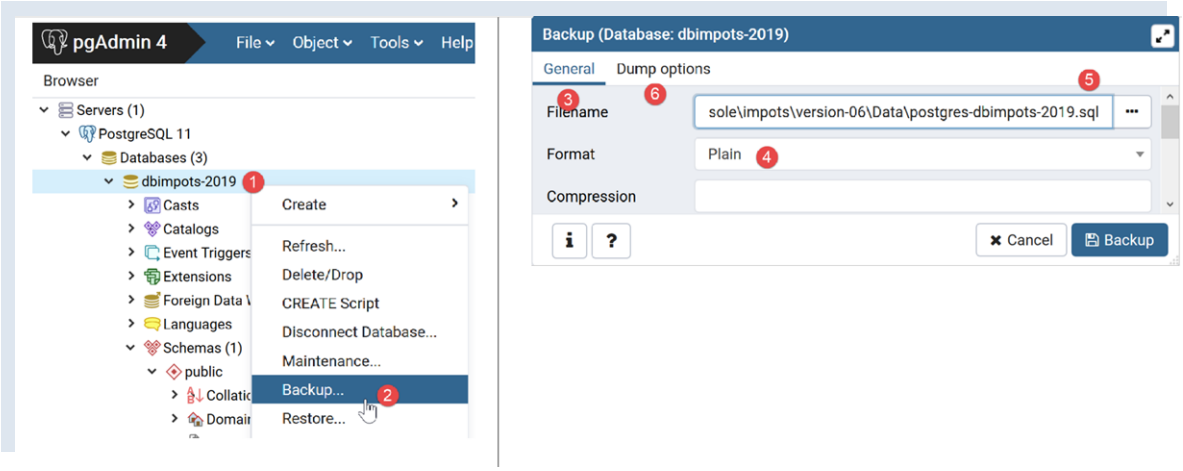


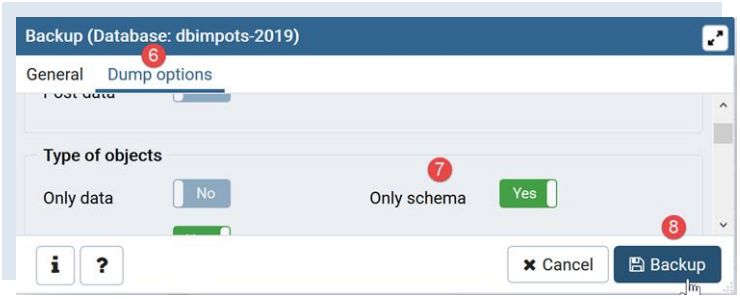




La base **[dbimpots-2019]** est désormais prête. Nous allons la remplir avec des données.

Comme nous l'avons fait avec MySQL, il est possible d'exporter la base de données **[dbimpots-2019]** dans un fichier SQL. On peut ensuite importer ce fichier SQL pour recréer la base si on l'a perdue ou détériorée. Nous n'exporterons ici que la structure de la base et non ses données :





Le fichier généré est le suivant :

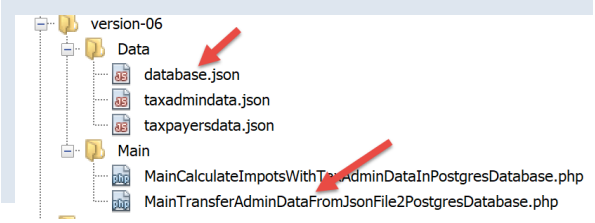
```
1   --
2   -- PostgreSQL database dump
3   --
4
5   -- Dumped from database version 11.2
6   -- Dumped by pg_dump version 11.2
7
8   -- Started on 2019-07-04 08:20:31
9
10  SET statement_timeout = 0;
11  SET lock_timeout = 0;
12  SET idle_in_transaction_session_timeout = 0;
13  SET client_encoding = 'UTF8';
14  SET standard_conforming_strings = on;
15  SELECT pg_catalog.set_config('search_path', '', false);
16  SET check_function_bodies = false;
17  SET client_min_messages = warning;
18  SET row_security = off;
19
20  SET default_tablespace = '';
21
22  SET default_with_oids = false;
23
24  --
25  -- TOC entry 198 (class 1259 OID 16408)
26  -- Name: tbconstantes; Type: TABLE; Schema: public; Owner: postgres
27  --
28
29  CREATE TABLE public.tbconstantes (
30      plafond_qf_demi_part double precision NOT NULL,
31      id integer NOT NULL,
32      plafond_revenus_celibataire_pour_reduction double precision NOT NULL,
33      plafond_revenus_couple_pour_reduction double precision NOT NULL,
34      valeur_reduc_demi_part double precision NOT NULL,
35      plafond_decote_celibataire double precision NOT NULL,
36      plafond_decote_couple double precision NOT NULL,
37      plafond_impot_celibataire_pour_decote double precision NOT NULL,
38      plafond_impot_couple_pour_decote double precision NOT NULL,
```

```
    abattement_dix_pourcent_max double precision NOT NULL,
    abattement_dix_pourcent_min double precision NOT NULL
);


ALTER TABLE public.tbconstantes OWNER TO postgres;

--
-- TOC entry 199 (class 1259 OID 16411)
-- Name: tbconstantes_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.tbconstantes ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY (
    SEQUENCE NAME public.tbconstantes_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 196 (class 1259 OID 16399)
-- Name: tbtranches; Type: TABLE; Schema: public; Owner: admimpots
--

CREATE TABLE public.tbtranches (
    limites double precision NOT NULL,
    id integer NOT NULL,
    coeffr double precision NOT NULL,
    coeffn double precision NOT NULL
);


ALTER TABLE public.tbtranches OWNER TO admimpots;

--
-- TOC entry 197 (class 1259 OID 16404)
-- Name: tbimpots_id_seq; Type: SEQUENCE; Schema: public; Owner: admimpots
--

ALTER TABLE public.tbtranches ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY (
    SEQUENCE NAME public.tbimpots_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 2694 (class 2606 OID 16429)
-- Name: tbconstantes tbconstantes_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.tbconstantes
    ADD CONSTRAINT tbconstantes_pkey PRIMARY KEY (id);


--
-- TOC entry 2692 (class 2606 OID 16403)
-- Name: tbtranches tbimpots_pkey; Type: CONSTRAINT; Schema: public; Owner: admimpots
--

ALTER TABLE ONLY public.tbtranches
    ADD CONSTRAINT tbimpots_pkey PRIMARY KEY (id);


--
-- TOC entry 2821 (class 0 OID 0)
-- Dependencies: 198
-- Name: TABLE tbconstantes; Type: ACL; Schema: public; Owner: postgres
--

GRANT ALL ON TABLE public.tbconstantes TO admimpots;
```

```
116
117
118 -- Completed on 2019-07-04 08:20:32
119
120 --
121 -- PostgreSQL database dump complete
122 --
```

1.14.4 Remplissage de la table `[tbtranches]`

Nous avons déjà fait ce travail avec le SGBD MySQL au paragraphe lien. Il nous suffit de modifier le fichier **[database.json]** qui décrit la base de données :



Le fichier **[database.json]** devient le suivant :

```
1   {
2       "dsn": "pgsql:host=localhost;dbname=dbimpots-2019",
3       "id": "admimpots",
4       "pwd": "mdpimpots",
5       "tableTranches": "public.tbtranches",
6       "colLimites": "limites",
7       "colCoeffR": "coeffr",
8       "colCoeffN": "coeffn",
9       "tableConstantes": "public.tbconstantes",
10      "colPlafondQfDemiPart": "plafond_qf_demi_part",
11      "colPlafondRevenusCelibatairePourReduction": "plafond_revenus_celibataire_pour_reduction",
12      "colPlafondRevenusCouplePourReduction": "plafond_revenus_couple_pour_reduction",
13      "colValeurReducDemiPart": "valeur_reduc_demi_part",
14      "colPlafondDecoteCelibataire": "plafond_decote_celibataire",
15      "colPlafondDecoteCouple": "plafond_decote_couple",
16      "colPlafondImpotCelibatairePourDecote": "plafond_impot_celibataire_pour_decote",
17      "colPlafondImpotCouplePourDecote": "plafond_impot_couple_pour_decote",
18      "colAbattementDixPourcentMax": "abattement_dix_pourcent_max",
19      "colAbattementDixPourcentMin": "abattement_dix_pourcent_min"
20  }
```

- ligne 2 : le DSN a changé, **[pgsql]** indiquant qu'on a affaire au SGBD Postgres ;
- lignes 5 et 9 : on a précédé le nom des tables par le nom du schéma auquel elles appartiennent **[public]**. Ce n'était pas indispensable puisque que **[public]** est le schéma utilisé par défaut lorsqu'aucun schéma n'est précisé dans le nom de la table ;
- lignes 6-8, 10-19 : les noms des colonnes ont changé ;

Le script **[MainTransferAdminDataFromJsonFile2PostgresDatabase.php]** de remplissage de la base **[dbimpots-2019]** est le suivant :

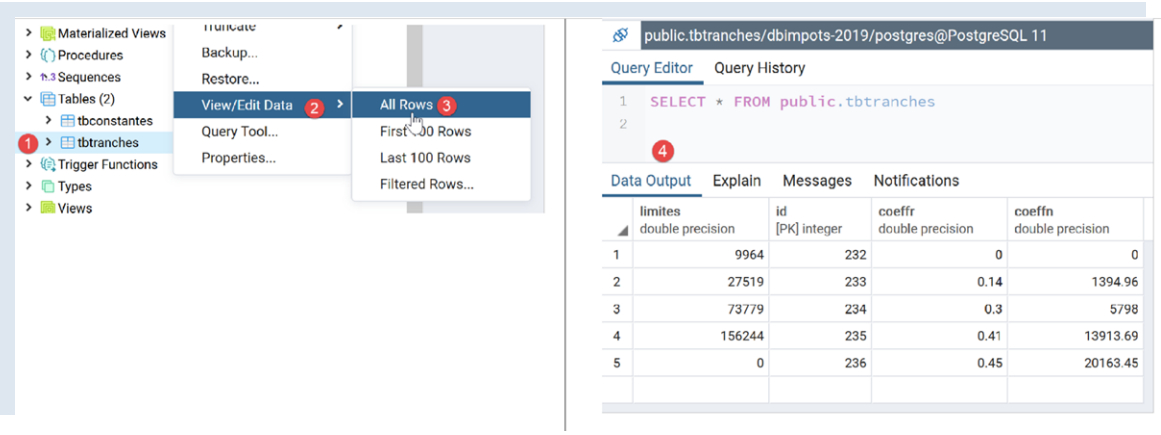
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // gestion des erreurs par PHP
10  // ini_set("display_errors", "0");
11  // inclusion interface et classes
12  require_once __DIR__ . "/../../version-05/Entities/BaseEntity.php";
13  require_once __DIR__ . "/../../version-05/Entities/TaxAdminData.php";
14  require_once __DIR__ . "/../../version-05/Entities/TaxPayerData.php";
15  require_once __DIR__ . "/../../version-05/Entities/Database.php";
16  require_once __DIR__ . "/../../version-05/Entities/ExceptionImpots.php";
17  require_once __DIR__ . "/../../version-05/Utilities/Utilitaires.php";
18  require_once __DIR__ . "/../../version-05/Dao/InterfaceDao.php";
```

```
19  require_once __DIR__ . "/../../version-05/Dao/TraitDao.php";
20  require_once __DIR__ . "/../../version-05/Dao/InterfaceDao4TransferAdminData2Database.php";
21  require_once __DIR__ . "/../../version-05/Dao/DaoTransferAdminDataFromJsonFile2Database.php";
22  //
23  // définition des constantes
24  const DATABASE_CONFIG_FILENAME = "../Data/database.json";
25  const TAXADMINDATA_FILENAME = "../Data/taxadmindata.json";
26
27  //
28  try {
29    // création de la couche [dao]
30    $dao = new DaoTransferAdminDataFromJsonFile2Database(DATABASE_CONFIG_FILENAME, TAXADMINDATA_FILENAME);
31    // transfert des données dans la base
32    $dao->transferAdminData2Database();
33  } catch (ExceptionImpots $ex) {
34    // on affiche l'erreur
35    print "L'erreur suivante s'est produite : " . utf8_encode($ex->getMessage()) . "\n";
36  }
37  // fin
38  print "Terminé\n";
39  exit;
```

Commentaires

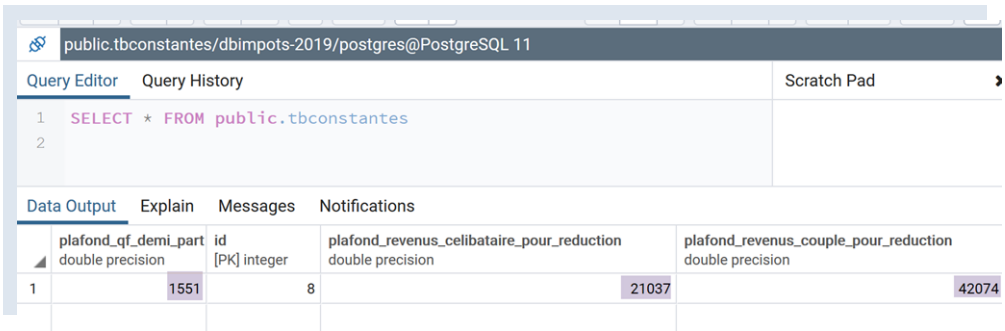
Seules les lignes 12-21 qui chargent les fichiers nécessaires à l'exécution de l'application changent. Elles changent parce que la valeur **[__DIR__]** change : elle désigne désormais le dossier **[version-07/Main]**.

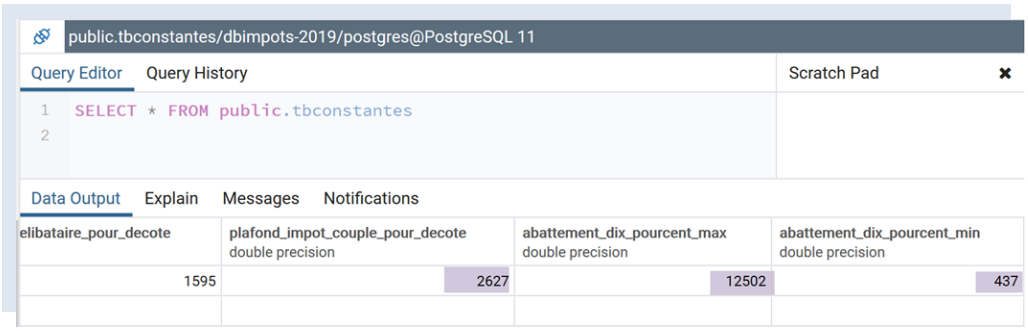
Lorsqu'on exécute ce script, on obtient le résultat suivant dans la table **[tbtranches]** :



- on clique droit sur **[1]**, puis ensuite **[2-3]** ;
- en **[4]**, on a bien les données des tranches d'impôts ;

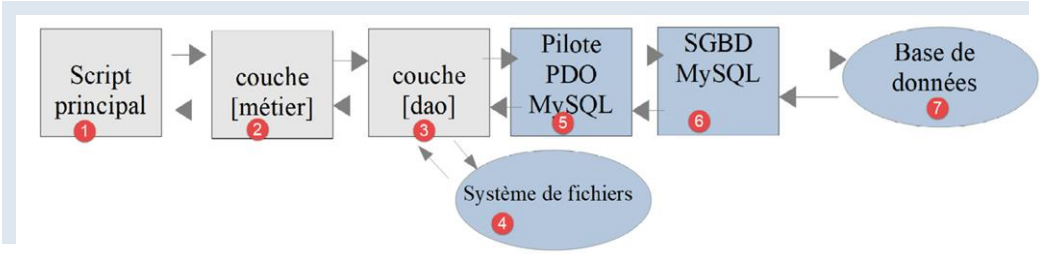
On refait la même chose pour la table des constantes **[tbconstantes]** :



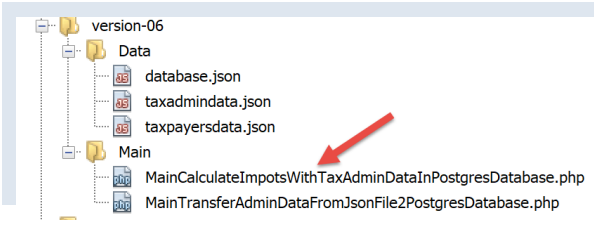


On notera que pour l'exécution du script, l'application Laragon n'a pas besoin d'être active : on n'a besoin ni du serveur Apache, ni du SGBD MySQL. On a seulement besoin du SGBD PostgreSQL dont on a lancé le service windows.

1.14.5 Calcul de l'impôt



Les couches **[dao]** (3) et **[métier]** (2) ont déjà été écrites. Nous avons déjà écrit le script principal pour le SGBD MySQL au paragraphe lien. Il nous suffit de reprendre le script **[MainCalculateImpotsWithTaxAdminDataInMySQLDatabase.php]** et de l'adapter au SGBD PostgreSQL. Il s'appelle désormais **[MainCalculateImpotsWithTaxAdminDataInPostgresDatabase.php]** :



Le script **[MainCalculateImpotsWithTaxAdminDataInPostgresDatabase.php]** est le suivant :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
```

```
9   // gestion des erreurs par PHP
10  //ini_set("display_errors", "0");
11  // inclusion interface et classes
12  require_once __DIR__ . "/../../version-05/Entities/BaseEntity.php";
13  require_once __DIR__ . "/../../version-05/Entities/TaxAdminData.php";
14  require_once __DIR__ . "/../../version-05/Entities/TaxPayerData.php";
15  require_once __DIR__ . "/../../version-05/Entities/Database.php";
16  require_once __DIR__ . "/../../version-05/Entities/ExceptionImpots.php";
17  require_once __DIR__ . "/../../version-05/Utilities/Utilitaires.php";
18  require_once __DIR__ . "/../../version-05/Dao/InterfaceDao.php";
19  require_once __DIR__ . "/../../version-05/Dao/TraitDao.php";
20  require_once __DIR__ . "/../../version-05/Dao/DaoImpotsWithTaxAdminDataInDatabase.php";
21  require_once __DIR__ . "/../../version-05/Métier/InterfaceMetier.php";
22  require_once __DIR__ . "/../../version-05/Métier/Metier.php";
23  //
24  // définition des constantes
25  const DATABASE_CONFIG_FILENAME = "../Data/database.json";
26  const TAXADMINDATA_FILENAME = "../Data/taxadmindata.json";
27  const RESULTS_FILENAME = "../Data/resultats.json";
28  const ERRORS_FILENAME = "../Data/errors.json";
29  const TAXPAYERSDATA_FILENAME = "../Data/taxpayersdata.json";
30
31  try {
32    // création de la couche [dao]
33    $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
34    // création de la couche [métier]
35    $métier = new Metier($dao);
36    // calcul de l'impôts en mode batch
37    $métier->executeBatchImpots(TAXPAYERSDATA_FILENAME, RESULTS_FILENAME, ERRORS_FILENAME);
38  } catch (ExceptionImpots $ex) {
39    // on affiche l'erreur
40    print "Une erreur s'est produite : " . utf8_encode($ex->getMessage()) . "\n";
41  }
42  // fin
43  print "Terminé\n";
44  exit;
```

Commentaires

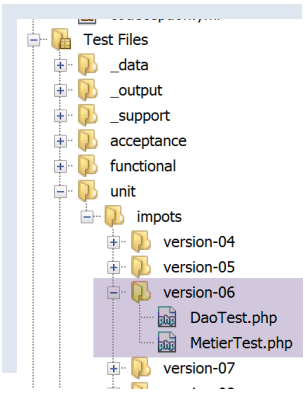
Seules les lignes 12-22 qui chargent les fichiers nécessaires à l'exécution de l'application changent. Elles changent parce que la valeur **[__DIR__]** change : elle désigne désormais le dossier **[version-07/Main]**.

Résultats d'exécution

Les mêmes que ceux obtenus dans les versions précédentes.

1.14.6 Tests [Codeception]

Comme pour les versions précédentes, nous validons cette version avec des tests **[Codeception]** :



1.14.6.1 Test de la couche [dao]

Le test **[DaoTest.php]** est le suivant :

```
1   <?php
2
```

```php
// respect strict des types déclarés des paramètres de foctions
declare (strict_types=1);

// espace de noms
namespace Application;

// répertoires racines
define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-06");
define("VENDOR", "C:/myprograms/laragon-lite/www/vendor");

// inclusion interface et classes
require_once ROOT . "/../version-05/Entities/BaseEntity.php";
require_once ROOT . "/../version-05/Entities/TaxAdminData.php";
require_once ROOT . "/../version-05/Entities/TaxPayerData.php";
require_once ROOT . "/../version-05/Entities/Database.php";
require_once ROOT . "/../version-05/Entities/ExceptionImpots.php";
require_once ROOT . "/../version-05/Utilities/Utilitaires.php";
require_once ROOT . "/../version-05/Dao/InterfaceDao.php";
require_once ROOT . "/../version-05/Dao/TraitDao.php";
require_once ROOT . "/../version-05/Dao/DaoImpotsWithTaxAdminDataInDatabase.php";

// bibliothèques tierces
require_once VENDOR . "/autoload.php";

// définition des constantes
const DATABASE_CONFIG_FILENAME = ROOT ."../Data/database.json";

class DaoTest extends \Codeception\Test\Unit {
  // TaxAdminData
  private $taxAdminData;

  public function __construct() {
    parent::__construct();
    // création de la couche [dao]
    $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
    $this->taxAdminData = $dao->getTaxAdminData();
  }

  // tests
  public function testTaxAdminData() {
    …
  }

}
```

Commentaires

- lignes 9-28 : définition de l'environnement du test. Nous utilisons le même, sans la couche **[métier]**, que celui utilisé par le script principal **[MainCalculateImpotsWithTaxAdminDataInPostgresDatabase]** décrit au paragraphe lien ;
- lignes 34-39 : construction de la couche **[dao]** ;
- ligne 38 : l'attribut **[$this→taxAdminData]** contient les données à tester ;
- lignes 42-44 : la méthode **[testTaxAdminData]** est celle décrite au paragraphe lien ;

Les résultats du test sont les suivants :

1.14.6.2 Test de la couche [métier]

Le test **[MetierTest.php]** est le suivant :

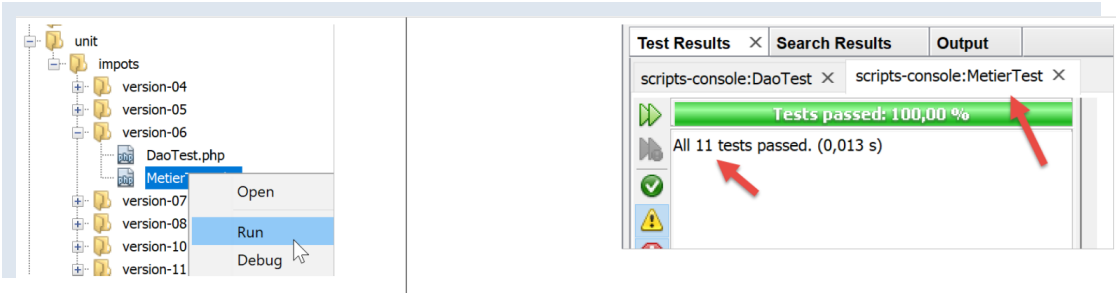
```php
<?php

```

```
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // répertoires racines
10  define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-06");
11  define("VENDOR", "C:/myprograms/laragon-lite/www/vendor");
12
13  // inclusion interface et classes
14  require_once ROOT . "/../version-05/Entities/BaseEntity.php";
15  require_once ROOT . "/../version-05/Entities/TaxAdminData.php";
16  require_once ROOT . "/../version-05/Entities/TaxPayerData.php";
17  require_once ROOT . "/../version-05/Entities/Database.php";
18  require_once ROOT . "/../version-05/Entities/ExceptionImpots.php";
19  require_once ROOT . "/../version-05/Utilities/Utilitaires.php";
20  require_once ROOT . "/../version-05/Dao/InterfaceDao.php";
21  require_once ROOT . "/../version-05/Dao/TraitDao.php";
22  require_once ROOT . "/../version-05/Dao/DaoImpotsWithTaxAdminDataInDatabase.php";
23  require_once ROOT . "/../version-05/Métier/InterfaceMetier.php";
24  require_once ROOT . "/../version-05/Métier/Metier.php";
25  // bibliothèques tierces
26  require_once VENDOR . "/autoload.php";
27  // définition des constantes
28  const DATABASE_CONFIG_FILENAME = ROOT . "../Data/database.json";
29
30  class MetierTest extends \Codeception\Test\Unit {
31    // couche métier
32    private $métier;
33
34    public function __construct() {
35      parent::__construct();
36      // création de la couche [dao]
37      $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
38      // création de la couche [métier]
39      $this->métier = new Metier($dao);
40    }
41
42    // tests
43    public function test1() {
44      …
45    }
46  ------------------------------------------------------------------------
47    public function test11() {
48      …
49    }
50
51  }
```

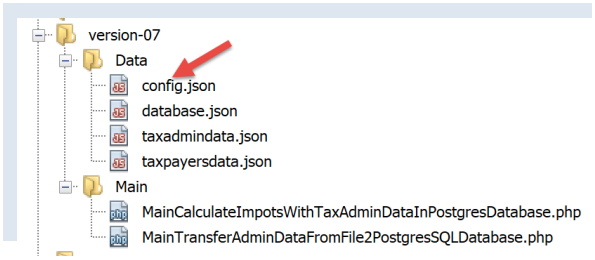
Commentaires

- lignes 9-28 : définition de l'environnement du test. Nous utilisons le même que celui utilisé par le script principal **[MainCalculateImpotsWithTaxAdminDataInPostgresDatabase]** décrit au paragraphe lien ;
- lignes 34-40 : construction des couches **[dao]** et **[métier]** ;
- ligne 39 : l'attribut **[$this->métier]** référence la couche **[métier]**
- lignes 43-49 : les méthodes **[test1, test2…, test11]** sont celles décrites au paragraphe lien ;

Les résultats du test sont les suivants :



1.15 Exercice d'application – version 7



1.15.1 Implémentation

Nous allons ici reprendre la version 6 en externalisant dans un fichier de configuration les constantes utilisées dans les scripts principaux. Le fichier de configuration sera un fichier jSON dont le contenu sera le suivant :

```
1   {
2       "rootDirectory": "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-07",
3       "databaseFilename": "Data/database.json",
4       "taxAdminDataFileName": "Data/taxadmindata.json",
5       "taxPayersDataFileName": "Data/taxpayersdata.json",
6       "resultsFileName": "Data/results.json",
7       "errorsFileName": "Data/errors.json",
8       "dependencies": {
9           "BaseEntity": "/../version-05/Entities/BaseEntity.php",
10          "TaxAdminData": "/../version-05/Entities/TaxAdminData.php",
11          "TaxPayerData": "/../version-05/Entities/TaxPayerData.php",
12          "Database": "/../version-05/Entities/Database.php",
13          "ExceptionImpots": "/../version-05/Entities/ExceptionImpots.php",
14          "Utilitaires": "/../version-05/Utilities/Utilitaires.php",
15          "InterfaceDao": "/../version-05/Dao/InterfaceDao.php",
16          "TraitDao": "/../version-05/Dao/TraitDao.php",
17          "InterfaceDao4TransferAdminData2Database": "/../version-05/Dao/InterfaceDao4TransferAdminData2Database.php",
18          "DaoTransferAdminDataFromJsonFile2Database": "/../version-05/Dao/DaoTransferAdminDataFromJsonFile2Database.php",
19          "DaoImpotsWithTaxAdminDataInDatabase": "/../version-05/Dao/DaoImpotsWithTaxAdminDataInDatabase.php",
20          "InterfaceMetier": "/../version-05/Métier/InterfaceMetier.php",
21          "Metier": "/../version-05/Métier/Metier.php"
22      },
23      "dependencies4calculate": [
24           "BaseEntity",
25           "TaxAdminData",
26           "TaxPayerData",
27           "Database",
28           "ExceptionImpots",
29           "Utilitaires",
30           "InterfaceDao",
31           "TraitDao",
32           "DaoImpotsWithTaxAdminDataInDatabase",
33           "InterfaceMetier",
34           "Metier"],
35      "dependencies4transfer": [
36           "BaseEntity",
37           "TaxAdminData",
38           "Database",
39           "ExceptionImpots",
40           "Utilitaires",
41           "InterfaceDao",
42           "TraitDao",
43           "InterfaceDao4TransferAdminData2Database",
44           "DaoTransferAdminDataFromJsonFile2Database"]
45  }
```

Dans cette configuration :

- ligne 2 : le dossier à partir duquel tous les chemins de ce fichier de configuration sont mesurés ;
- lignes 3-7 : les chemins de tous les fichiers jSON de l'application ;
- lignes 8-22 : les chemins de tous les fichiers de l'application, sous la formé **`[clé=>chemin]`** ;
- lignes 23-34 : les dépendances pour le calcul de l'impôt sous la forme d'une liste de clés du dictionnaire **`[dependencies]`** (lignes 8-22) ;

- lignes 35-44 : les dépendances pour le transfert en base des données du fichier jSON **[taxadmindata.json]** sous la forme d'une liste de clés du dictionnaire **[dependencies]** (lignes 8-22) ;

Le script de transfert en base des données de l'administration fiscale **[MainTransferAdminDataFromFile2PostgresSQLDatabase.php]** devient le suivant :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // gestion des erreurs par PHP
10  // ini_set("display_errors", "0");
11  //
12  // chemin du fichier de configuration
13  define("CONFIG_FILENAME", "../Data/config.json");
14
15  // on récupère la configuration
16  $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
17
18  // on inclut les dépendances nécessaires au script
19  $rootDirectory = $config["rootDirectory"];
20  foreach ($config["dependencies4transfer"] as $dependency) {
21    require $rootDirectory . $config["dependencies"][$dependency];
22  }
23
24  // définition des constantes
25  define("DATABASE_CONFIG_FILENAME", "$rootDirectory/{$config["databaseFilename"]}");
26  define("TAXADMINDATA_FILENAME", "$rootDirectory/{$config["taxAdminDataFileName"]}");
27
28  //
29  try {
30    // création de la couche [dao]
31    $dao = new DaoTransferAdminDataFromJsonFile2Database(DATABASE_CONFIG_FILENAME, TAXADMINDATA_FILENAME);
32    // transfert des données dans la base
33    $dao->transferAdminData2Database();
34  } catch (ExceptionImpots $ex) {
35    // on affiche l'erreur
36    print $ex->getMessage() . "\n";
37  }
38  // fin
39  print "Terminé\n";
40  exit;
```

Commentaires

Le code reste celui qu'il était au paragraphe lien. La seule différence est l'exploitation du fichier **[config.json]** en lieu et place des constantes aux lignes 18-26 ;

- ligne 16 : la fonction **[file_get_contents]** transfère le fichier **[config.json]** dans une chaîne de caractères. La fonction **[json_decode]** exploite ensuite cette chaîne pour construire le dictionnaire **[$config]**. Le second paramètre **[true]** de la fonction **[json_decode]** indique qu'on veut construire un dictionnaire ;
- lignes 19-22 : on inclut les dépendances nécessaires au script de transfert des données du fichier **[taxadmindata.json]** vers la base de données ;
  - **[$config["dependencies4transfer"]]** est le tableau des dépendances nécessaires au script de transfert. C'est une liste de **clés**. Les chemins des fichiers à inclure dans le projet sont trouvés dans le dictionnaire **[$config["dependencies"]]** ;
  - `$config`**["rootDirectory"]** représente le chemin avec lequel les fichiers à inclure doivent être préfixés ;

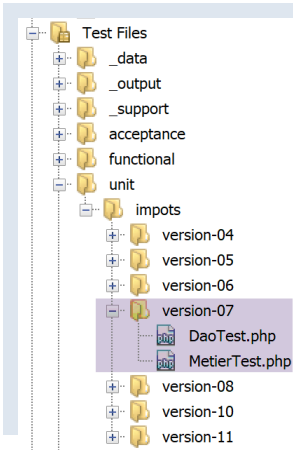
De la même façon, le script de calcul de l'impôt devient le suivant **[MainCalculateImpotsWithTaxAdminDataInPostgresDatabase]** :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // gestion des erreurs par PHP
10  // ini_set("display_errors", "0");
```

```php
//
// chemin du fichier de configuration
define("CONFIG_FILENAME", "../Data/config.json");

// on récupère la configuration
$config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);

// on inclut les dépendances nécessaires au script
$rootDirectory = $config["rootDirectory"];
foreach ($config["dependencies4calculate"] as $dependency) {
  require $rootDirectory . $config["dependencies"][$dependency];
}

// définition des constantes
define("DATABASE_CONFIG_FILENAME", "$rootDirectory/{$config["databaseFilename"]}");
define("TAXPAYERSDATA_FILENAME", "$rootDirectory/{$config["taxPayersDataFileName"]}");
define("RESULTS_FILENAME", "$rootDirectory/{$config["resultsFileName"]}");
define("ERRORS_FILENAME", "$rootDirectory/{$config["errorsFileName"]}");

//
try {
  // création de la couche [dao]
  $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
  // création de la couche [métier]
  $métier = new Metier($dao);
  // calcul de l'impôts en mode batch
  $métier->executeBatchImpots(TAXPAYERSDATA_FILENAME, RESULTS_FILENAME, ERRORS_FILENAME);
} catch (ExceptionImpots $ex) {
  // on affiche l'erreur
  print "Une erreur s'est produite : " . utf8_encode($ex->getMessage()) . "\n";
}
// fin
print "Terminé\n";
exit;
```

1.15.2 Tests [Codeception]

Cette version comme les précédentes est validée par des tests **[Codeception]**.



1.15.2.1 Test de la couche [dao]

Le test **[DaoTest.php]** est le suivant :

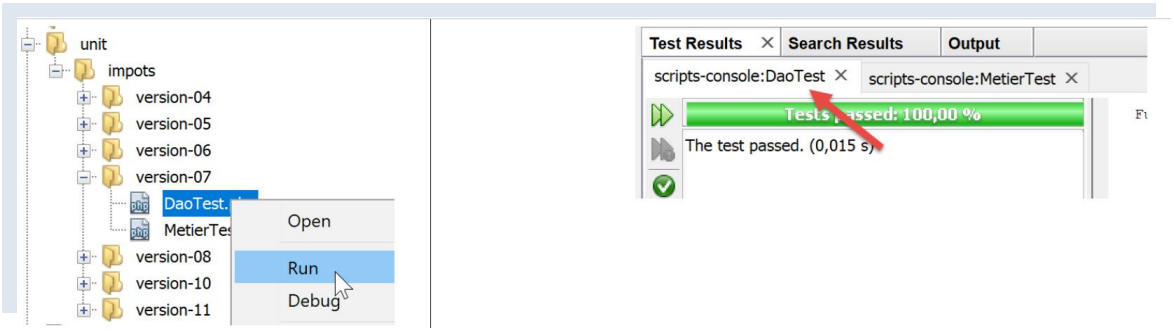
```php
<?php

// respect strict des types déclarés des paramètres de fonctions
declare (strict_types=1);

// espace de noms
namespace Application;

// définition de constantes
define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-07");
// chemin du fichier de configuration
define("CONFIG_FILENAME", ROOT."/Data/config.json");
```

```
13  // on récupère la configuration
14  $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
15  // on inclut les dépendances nécessaires au script
16  $rootDirectory = $config["rootDirectory"];
17  foreach ($config["dependencies4calculate"] as $dependency) {
18    require $rootDirectory . $config["dependencies"][$dependency];
19  }
20  // autres constantes
21  define("DATABASE_CONFIG_FILENAME", "$rootDirectory/{$config["databaseFilename"]}");
22
23  // test ----------------------------------------------------
24
25  class DaoTest extends \Codeception\Test\Unit {
26    // TaxAdminData
27    private $taxAdminData;
28
29    public function __construct() {
30      parent::__construct();
31      // création de la couche [dao]
32      $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
33      $this->taxAdminData = $dao->getTaxAdminData();
34    }
35
36    // tests
37    public function testTaxAdminData() {
38      …
39    }
40
41  }
```

Commentaires

- lignes 9-21 : définition de l'environnement du test. Nous utilisons le même, sans la couche **[métier]**, que celui utilisé par le script principal **[MainCalculateImpotsWithTaxAdminDataInPostgresDatabase]** décrit au paragraphe lien ;
- lignes 29-34 : construction de la couche **[dao]** ;
- ligne 33 : l'attribut **[$this→taxAdminData]** contient les données à tester ;
- lignes 37-39 : la méthode **[testTaxAdminData]** est celle décrite au paragraphe lien ;

Les résultats du test sont les suivants :



1.15.2.2 Test de la couche [métier]

Le test **[MetierTest.php]** est le suivant :

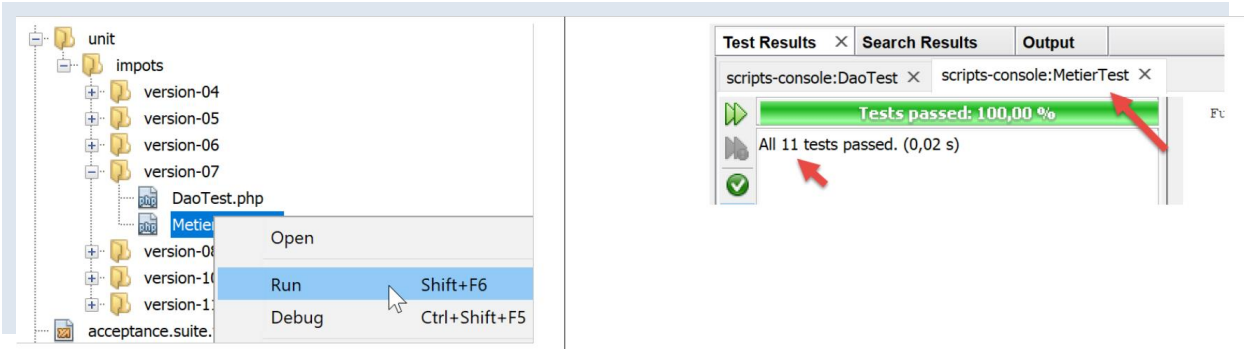
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // définition de constantes
10  define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-07");
11  // chemin du fichier de configuration
12  define("CONFIG_FILENAME", ROOT . "/Data/config.json");
13  // on récupère la configuration
14  $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
15  // on inclut les dépendances nécessaires au script
```

```
16  $rootDirectory = $config["rootDirectory"];
17  foreach ($config["dependencies4calculate"] as $dependency) {
18    require $rootDirectory . $config["dependencies"][$dependency];
19  }
20  // autres constantes
21  define("DATABASE_CONFIG_FILENAME", "$rootDirectory/{$config["databaseFilename"]}");
22
23  // classe de test
24  class MetierTest extends \Codeception\Test\Unit {
25    // couche métier
26    private $métier;
27
28    public function __construct() {
29      parent::__construct();
30      // création de la couche [dao]
31      $dao = new DaoImpotsWithTaxAdminDataInDatabase(DATABASE_CONFIG_FILENAME);
32      // création de la couche [métier]
33      $this->métier = new Metier($dao);
34    }
35
36    // tests
37    public function test1() {
38      …
39    }
40
41    ----------------------------------
42
43    public function test11() {
44      …
45    }
46
47  }
```

Commentaires

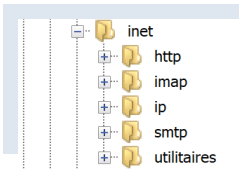
- lignes 9-21 : définition de l'environnement du test. Nous utilisons le même que celui utilisé par le script principal **[MainCalculateImpotsWithTaxAdminDataInPostgresDatabase]** décrit au paragraphe lien ;
- lignes 28-34 : construction de la couche **[dao]** ;
- ligne 33 : l'attribut **[$this->métier]** est une référence sur la couche **[métier]** à tester ;
- lignes 37-45 : les méthodes **[test1, test2…, test11]** sont celles décrites au paragraphe lien ;

Les résultats du test sont les suivants :



1.16 Fonctions réseau

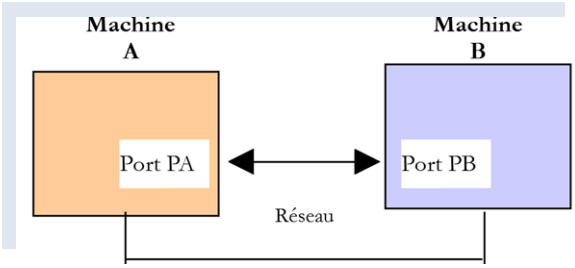
Nous abordons maintenant les fonctions réseau de PHP qui nous permettent de faire de la programmation TCP / IP (**T**ransfer **C**ontrol **P**rotocol / **I**nternet **P**rotocol).



1.16.1 Les bases de la programmation internet

1.16.1.1 Généralités

Considérons la communication entre deux machines distantes A et B :



Lorsque une application *AppA* d'une machine A veut communiquer avec une application *AppB* d'une machine B de l'Internet, elle doit connaître plusieurs choses :

- l'adresse IP (**I**nternet **P**rotocol) ou le nom de la machine B ;
- le numéro du port avec lequel travaille l'application *AppB*. En effet la machine B peut supporter de nombreuses applications qui travaillent sur l'Internet. Lorsqu'elle reçoit des informations provenant du réseau, elle doit savoir à quelle application sont destinées ces informations. Les applications de la machine B ont accès au réseau via des guichets appelés également des **ports de communication**. Cette information est contenue dans le paquet reçu par la machine B afin qu'il soit délivré à la bonne application ;
- les protocoles de communication compris par la machine B. Dans notre étude, nous utiliserons uniquement les protocoles TCP-IP ;
- le protocole de dialogue accepté par l'application *AppB*. En effet, les machines A et B vont se "parler". Ce qu'elles vont dire va être encapsulé dans les protocoles TCP-IP. Néanmoins, lorsqu'au bout de la chaîne, l'application *AppB* va recevoir l'information envoyée par l'applicaton *AppA*, il faut qu'elle soit capable de l'interpréter. Ceci est analogue à la situation où deux personnes A et B communiquent par téléphone : leur dialogue est transporté par le téléphone. La parole va être codée sous forme de signaux par le téléphone A, transportée par des lignes téléphoniques, arriver au téléphone B pour y être décodée. La personne B entend alors des paroles. C'est là qu'intervient la notion de protocole de dialogue : si A parle français et que B ne comprend pas cette langue, A et B ne pourront dialoguer utilement ;

Aussi les deux applications communicantes doivent-elles être d'accord sur le type de dialogue qu'elles vont adopter. Par exemple, le dialogue avec un service *ftp* n'est pas le même qu'avec un service *pop* : ces deux services n'acceptent pas les mêmes commandes. Elles ont un protocole de dialogue différent ;

1.16.1.2 Les caractéristiques du protocole TCP

Nous n'étudierons ici que des communications réseau utilisant le protocole de transport TCP dont voici les principales caractéristiques :

- le processus qui souhaite émettre établit tout d'abord une **connexion** avec le processus destinataire des informations qu'il va émettre. Cette connexion se fait entre un port de la machine émettrice et un port de la machine réceptrice. Il y a entre les deux ports un chemin virtuel qui est ainsi créé et qui sera réservé aux deux seuls processus ayant réalisé la connexion ;
- tous les paquets émis par le processus source suivent ce chemin virtuel et arrivent dans l'ordre où ils ont été émis ;
- l'information émise a un aspect continu. Le processus émetteur envoie des informations à son rythme. Celles-ci ne sont pas nécessairement envoyées tout de suite : le protocole TCP attend d'en avoir assez pour les envoyer. Elles sont stockées dans une structure appelée *segment TCP*. Ce segment une fois rempli sera transmis à la couche IP où il sera encapsulé dans un paquet IP ;
- chaque segment envoyé par le protocole TCP est numéroté. Le protocole TCP destinataire vérifie qu'il reçoit bien les segments en séquence. Pour chaque segment correctement reçu, il envoie un accusé de réception à l'expéditeur ;
- lorsque ce dernier le reçoit, il l'indique au processus émetteur. Celui-ci peut donc savoir qu'un segment est arrivé à bon port ;
- si au bout d'un certain temps, le protocole TCP ayant émis un segment ne reçoit pas d'accusé de réception, il retransmet le segment en question, garantissant ainsi la qualité du service d'acheminement de l'information ;
- le circuit virtuel établi entre les deux processus qui communiquent est *full-duplex* : cela signifie que l'information peut transiter dans les deux sens. Ainsi le processus destination peut envoyer des accusés de réception alors même que le processus source

continue d'envoyer des informations. Cela permet par exemple au protocole TCP source d'envoyer plusieurs segments sans attendre d'accusé de réception. S'il réalise au bout d'un certain temps qu'il n'a pas reçu l'accusé de réception d'un certain segment n° *n*, il reprendra l'émission des segments à ce point ;

1.16.1.3 La relation client-serveur

Souvent, la communication sur Internet est dissymétrique : la machine A initie une connexion pour demander un service à la machine B : il précise qu'il veut ouvrir une connexion avec le service SB1 de la machine B. Celle-ci accepte ou refuse. Si elle accepte, la machine A peut envoyer ses demandes au service SB1. Celles-ci doivent se conformer au protocole de dialogue compris par le service SB1. Un dialogue demande-réponse s'instaure ainsi entre la machine A qu'on appelle machine **cliente** et la machine B qu'on appelle machine **serveur**. L'un des deux partenaires fermera la connexion.

1.16.1.4 Architecture d'un client

L'architecture d'un programme réseau demandant les services d'une application serveur sera la suivante :

```
ouvrir la connexion avec le service SB1 de la machine B
si réussite alors
    tant que ce n'est pas fini
            préparer une demande
            l'émettre vers la machine B
            attendre et récupérer la réponse
            la traiter
    fin tant que
finsi
fermer la connexion
```

1.16.1.5 Architecture d'un serveur

L'architecture d'un programme offrant des services sera la suivante :

```
ouvrir le service sur la machine locale
tant que le service est ouvert
    se mettre à l'écoute des demandes de connexion sur un port dit port d'écoute
    lorsqu'il y a une demande, la faire traiter par une autre tâche sur un autre port dit port de
service
fin tant que
```

Le programme serveur traite différemment la demande de connexion initiale d'un client de ses demandes ultérieures visant à obtenir un service. Le programme n'assure pas le service lui-même. S'il le faisait, pendant la durée du service il ne serait plus à l'écoute des demandes de connexion et des clients ne seraient alors pas servis. Il procède donc autrement : dès qu'une demande de connexion est reçue sur le port d'écoute puis acceptée, le serveur crée une tâche chargée de rendre le service demandé par le client. Ce service est rendu sur un autre port de la machine serveur appelé **port de service**. On peut ainsi servir plusieurs clients en même temps.

Une tâche de service aura la structure suivante :

```
tant que le service n'a pas été rendu totalement
            attendre une demande sur le port de service
            lorsqu'il y en a une, élaborer la réponse
            transmettre la réponse via le port de service
fin tant que
libérer le port de service
```

1.16.2 Découvrir les protocoles de communication de l'internet

1.16.2.1 Introduction

Lorsqu'un client s'est connecté à un serveur, s'établit ensuite un dialogue entre-eux. La nature de celui-ci forme ce qu'on appelle le protocole de communication du serveur. Parmi les protocoles les plus courants de l'internet on trouve les suivants :

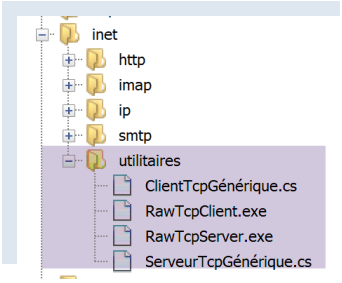
- HTTP : **H**yper**T**ext **T**ransfer **P**rotocol - le protocole de dialogue avec un serveur web (serveur HTTP) ;
- SMTP : **S**imple **M**ail **T**ransfer **P**rotocol - le protocole de dialogue avec un serveur d'envoi de courriers électroniques (serveur SMTP) ;
- POP : **P**ost **O**ffice **P**rotocol - le protocole de dialogue avec un serveur de stockage du courrier électronique (serveur POP). Il s'agit là de récupérer les courriers électroniques reçus et non d'en envoyer ;
- IMAP : **I**nternet **M**essage **A**ccess **P**rotocol - le protocole de dialogue avec un serveur de stockage du courrier électronique (serveur IMAP). Ce protocole a remplacé progressivement le protocole POP plus ancien ;
- FTP : **F**ile **T**ransfer **P**rotocol - le protocole de dialogue avec un serveur de stockage de fichiers (serveur FTP) ;

Tous ces protocoles ont la particularité d'être des protocoles à lignes de texte : le client et le serveur s'échangent des lignes de texte. Si on a un client capable de :

- créer une connexion avec un serveur TCP ;
- afficher à la console les lignes de texte que le serveur lui envoie ;
- envoyer au serveur les lignes de texte qu'un utilisateur saisirait au clavier ;

alors on est capable de dialoguer avec un serveur TCP ayant un protocole à lignes de texte pour peu qu'on connaisse les règles de ce protocole.

1.16.2.2 Utilitaires TCP



Dans les codes associés à ce document, on trouve deux utilitaires de communication TCP :

- **[RawTcpClient]** permet de se connecter sur le port P d'un serveur S ;
- **[RawTcpServer]** permet de créer un serveur qui attend des clients sur un port P ;

Le serveur TCP **[RawTcpServer]** s'appelle avec la syntaxe **[RawTcpServeur port]** pour créer un service TCP sur le port **[port]** de la machine locale (l'ordinateur sur lequel vous travaillez) :

- le serveur peut servir plusieurs clients simultanément ;
- le serveur exécute les commandes tapées par l'utilisateur tapées au clavier. Celles-ci sont les suivantes :
  - **list** : liste les clients actuellement connectés au serveur. Ceux-ci sont affichés sous la forme **[id=x-nom=y]**. Le champ **[id]** sert à identifier les clients ;
  - **send x [texte]** : envoie **texte** au client n° **x** (id=x). Les crochets **[]** ne sont pas envoyés. Ils sont nécessaires dans la commande. Ils servent à délimiter visuellement le texte envoyé au client ;
  - **close x** : ferme la connexion avec le client n° **x** ;
  - **quit** : ferme toutes les connexions et arrête le service ;
- les lignes envoyées par le client au serveur sont affichées sur la console ;
- l'ensemble des échanges est logué dans un fichier texte portant le nom **[machine-portService.txt]** où
  - **[machine]** est le nom de la machine sur laquelle s'exécute le code ;
  - **[port]** est le port de service qui répond aux demandes du client ;

Le client TCP **[RawTcpClient]** s'appelle avec la syntaxe **[RawTcpClient serveur port]** pour se connecter au port **[port]** du serveur **[serveur]** :

- les lignes tapées par l'utilisateur au clavier sont envoyées au serveur ;
- les lignes envoyées par le serveur sont affichées sur la console ;
- l'ensemble des échanges est logué dans un fichier texte portant le nom **[serveur-port.txt]** ;

Voyons un exemple. On ouvre deux fenêtres de commandes Windows et on se positionne dans chacune d'elles sur le dossier des utilitaires. Dans l'une des fenêtres on lance le serveur **[RawTcpServer]** sur le port 100 :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpServer.exe 100
server : Serveur générique lancé sur le port 0.0.0.0:100 1                    2
server : Attente d'un client... 3
Commandes disponibles : [list, send id [texte], close id, quit]
user : 4
```

- en **[1]**, nous sommes placés dans le dossier des utilitaires ;
- en **[2]**, nous lançons le serveur TCP sur le port 100 ;
- en **[3]**, le serveur se met en attente d'un client TCP ;
- en **[4]**, le serveur attend une commande tapée par l'utilisateur au clavier ;

Dans l'autre fenêtre de commandes, on lance le client TCP :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpClient localhost 100
Client [DESKTOP-528I5CU-50405] connecté au serveur [localhost-100]
Tapez vos commandes (quit pour arrêter) :
```

- en **[5]**, nous sommes placés dans le dossier des utilitaires ;
- en **[6]**, nous lançons le client TCP : nous lui disons de se connecter au port 100 de la machine locale (celle avec laquelle vous travaillez) ;
- en **[7]**, le client a réussi à se connecter au serveur. On indique les coordonnées du client : il est sur la machine **[DESKTOP-528I5CU]** (la machine locale dans cet exemple) et utilise le port **[50405]** pour communiquer avec le serveur :
- en **[8]**, le client attend une commande tapée par l'utilisateur au clavier ;

Revenons sur la fenêtre du serveur. Son contenu a évolué :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpServer.exe 100
server : Serveur générique lancé sur le port 0.0.0.0:100
server : Attente d'un client...
Commandes disponibles : [list, send id [texte], close id, quit]
user : server : Client 1-DESKTOP-528I5CU-50405 connecté...
server : Attente d'un client...
```

- en **[9]**, un client a été détecté. Le serveur lui a donné le n° 1. Le serveur a correctement identifié le client distant (machine et port) ;
- en **[10]**, le serveur se remet en attente d'un nouveau client ;

Revenons sur la fenêtre du client et envoyons une commande au serveur :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpClient localhost 100
Client [DESKTOP-528I5CU-50405] connecté au serveur [localhost-100]
Tapez vos commandes (quit pour arrêter) :
hello from client
```

- en **[11]**, la commande envoyée au serveur ;

Revenons sur la fenêtre du serveur. Son contenu a évolué :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpServer
server : Serveur générique lancé sur le port 0.0.0.0:100
server : Attente d'un client...
Commandes disponibles : [list, send id [texte], close id, quit]
user : server : Client 1-DESKTOP-528I5CU-50405 connecté...
server : Attente d'un client...
client 1 : [hello from client]
```

- en **[12]**, entre cochets, le message reçu par le serveur ;

Envoyons une réponse au client :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpS
server : Serveur générique lancé sur le port 0.0.0.0:100
server : Attente d'un client...
Commandes disponibles : [list, send id [texte], close id, quit]
user : server : Client 1-DESKTOP-528I5CU-50405 connecté...
server : Attente d'un client...
client 1 : [hello from client]
send 1 [hello from server]
user :
```

- en **[13]**, la réponse envoyée au client 1. Seul le texte entre les crochets est envoyé, pas les crochets eux-mêmes ;

Revenons à la fenêtre du client :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawT
Client [DESKTOP-528I5CU-50405] connecté au serveur [localhost-100]
Tapez vos commandes (quit pour arrêter) :
hello from client
<-- [hello from server] 14
```

- en **[14]**, la réponse reçue par le client. Le texte reçu est celui entre crochets ;

Revenons à la fenêtre du serveur pour voir d'autres commandes :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\util
server : Serveur générique lancé sur le port 0.0.0.0:100
server : Attente d'un client...
Commandes disponibles : [list, send id [texte], close id,
user : server : Client 1-DESKTOP-528I5CU-50405 connecté..
server : Attente d'un client...
client 1 : [hello from client]
send 1 [hello from server]
user : list 15
server : id=1-name=DESKTOP-528I5CU-50405 16
user : close 1 17
server : Connexion client 1 fermée... 18
user : quit 19
server : fin du service 20
```

- en **[15]**, nous demandons la liste des clients ;
- en **[16]**, la réponse ;
- en **[17]**, nous fermons la connexion avec le client n° 1 ;
- en **[18]**, la confirmation du serveur ;
- en **[19]**, nous arrêtons le serveur ;
- en **[20]**, la confirmation du serveur ;

Revenons à la fenêtre du client :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>Ra
Client [DESKTOP-528I5CU-50405] connecté au serveur [localhost-100]
Tapez vos commandes (quit pour arrêter) :
hello from client
<-- [hello from server]
Perte de la connexion avec le serveur... 21
```

- en **[21]**, le client a détecté la fin du service ;

Deux fichiers de logs ont été créés, un pour le serveur, un autre pour le client :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>dir 24
 Le volume dans le lecteur C s'appelle Local Disk 23
 Le numéro de série du volume est 2C89-ED9D

 Répertoire de C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires

16/05/2019  14:48    <DIR>          .
16/05/2019  14:48    <DIR>          ..
16/05/2019  14:43                50 DESKTOP-528I5CU-50405.txt 25
16/05/2019  14:43                50 localhost-100.txt 26
16/05/2019  14:26             7 168 RawTcpClient.exe
16/05/2019  14:27            10 240 RawTcpServer.exe
```

- en **[25]**, les logs du serveur : le nom du fichier est le nom du client **[machine-port]** ;
- en **[26]**, les logs du client : le nom du fichier est le nom du serveur **[machine-port]** ;

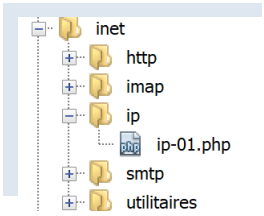
Les logs du serveur sont les suivants :

```
1   <-- [hello from client]
2   --> [hello from server]
```

Les logs du client sont les suivants :

```
1   --> [hello from client]
2   <-- [hello from server]
```

1.16.3 Obtenir le nom ou l'adresse IP d'une machine de l'Internet



Les machines de l'internet sont identifiées par une adresse IP (IPv4 ou IPv6) et le plus souvent par un nom. Mais finalement seule l'adresse IP est utilisée. Il faut donc parfois connaître l'adresse IP d'une machine identifiée par son nom.

Le script **[ip-01.php]** est le suivant :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de fonctions
4   declare (strict_types=1);
5   //
6   // gestion des erreurs
7   error_reporting(E_ALL & E_STRICT);
8   ini_set("display_errors", "on");
9   //
10  // constantes
11  $HOTES = array("istia.univ-angers.fr", "www.univ-angers.fr", "www.ibm.com", "localhost", "", "xx");
12  // adresses IP et noms des machines de $HOTES
13  for ($i = 0; $i < count($HOTES); $i++) {
14    getIPandName($HOTES[$i]);
15  }
16  // fin
17  print "Terminé\n";
18  exit;
19
20  //-----------------------------------------------
21  function getIPandName(string $nomMachine): void {
22    //$nomMachine : nom de la machine dont on veut l'adresse IP
23    //
24    // nomMachine-->adresse IP
25    $ip = gethostbyname($nomMachine);
26    print "---------------\n";
27    if ($ip !== $nomMachine) {
28      print "ip[$nomMachine]=$ip\n";
29      // adresse IP --> nomMachine
30      $name = gethostbyaddr($ip);
31      if ($name !== $ip) {
32        print "name[$ip]=$name\n";
33      } else {
34        print "Erreur, machine[$ip] non trouvée\n";
35      }
36    } else {
37      print "Erreur, machine[$nomMachine] non trouvée\n";
38    }
39  }
```

Commentaires

- lignes 7-8 : on demande à ce que PHP signale toutes les erreurs (E_ALL & E_STRICT) et que celles-ci soient affichées. Ce mode n'est recommandé qu'en mode développement pour améliorer le code avec les avertissements de PHP. En mode production, ligne 8, on mettrait « off ». Depuis PHP 5.4, le niveau E_STRICT est inclus dans E_ALL ;
- ligne 11 : la liste de machines dont on veut le nom et l'adresse IP ;

Les fonctions réseau de PHP sont utilisées dans la fonction *getIpandName* de la ligne 21.

- ligne 25 : la fonction *gethostbyname($nom)* permet d'obtenir l'adresse IP "ip3.ip2.ip1.ip0" de la machine s'appelant $*nom*. Si la machine $*nom* n'existe pas, la fonction rend $*nom* comme résultat ;

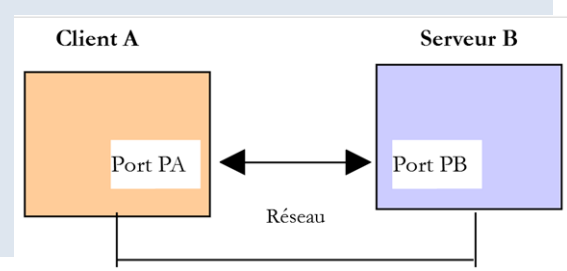
- ligne 30 : la fonction *gethostbyaddr($ip)* permet d'obtenir le nom de la machine d'adresse $ip de la forme "ip3.ip2.ip1.ip0". Si la machine $*ip* n'existe pas, la fonction rend $*ip* comme résultat ;

Résultats :

```
1   ---------------
2   ip[istia.univ-angers.fr]=193.49.144.41
3   name[193.49.144.41]=ametys-fo-2.univ-angers.fr
4   ---------------
5   ip[www.univ-angers.fr]=193.49.144.41
6   name[193.49.144.41]=ametys-fo-2.univ-angers.fr
7   ---------------
8   ip[www.ibm.com]=2.18.220.211
9   name[2.18.220.211]=a2-18-220-211.deploy.static.akamaitechnologies.com
10  ---------------
11  ip[localhost]=127.0.0.1
12  name[127.0.0.1]=DESKTOP-528I5CU
13  ---------------
14  ip[]=192.168.1.38
15  name[192.168.1.38]=DESKTOP-528I5CU.home
16  ---------------
17  Erreur, machine[xx] non trouvée
18  Terminé
```

1.16.4 Le protocole HTTP (HyperText Transfer Protocol)

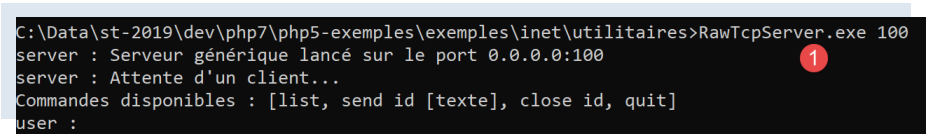
1.16.4.1 Exemple 1



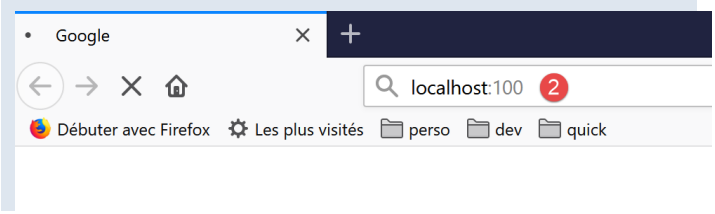
Lorsqu'un navigateur affiche une URL, il est le client d'un serveur web ou dit autrement d'un serveur HTTP. C'est lui qui prend l'initiative et il commence par envoyer un certain nombre de commandes au serveur. Pour ce premier exemple :

- le serveur sera l'utilitaire **[RawTcpServer]** ;
- le client sera un navigateur ;

Nous lançons d'abord le serveur sur le port 100 :



Puis avec un navigateur, nous demandons l'URL **[localhost:100]**, ç-a-d que nous disons que le serveur HTTP interrogé travaille sur le port 100 de la machine locale :



Revenons sur la fenêtre du serveur :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpServer.exe 100
server : Serveur générique lancé sur le port 0.0.0.0:100
server : Attente d'un client...
Commandes disponibles : [list, send id [texte], close id, quit]
user : server : Client 1-DESKTOP-528I5CU-50509 connecté...3
server : Attente d'un client...
client 1 : [GET / HTTP/1.1]4
client 1 : [Host: localhost:100]5
client 1 : [User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0]
client 1 : [Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8]
client 1 : [Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3]
client 1 : [Accept-Encoding: gzip, deflate]6
client 1 : [DNT: 1]
client 1 : [Connection: keep-alive]
client 1 : [Upgrade-Insecure-Requests: 1]
client 1 : []7
```

- en **[3]**, le client qui s'est connecté ;
- en **[4-7]**, la série de lignes de texte qu'il a envoyées :
  - en **[4]** : cette ligne a le format **[GET URL HTTP/1.1]**. Elle demande l'URL / et demande au serveur d'utiliser le protocole HTTP 1.1 ;
  - en **[5]** : cette ligne a le format **[Host: serveur:port]**. La casse de la commande **[Host]** n'importe pas. On rappelle ici que le client interroge un serveur local opérant sur le port 100 ;
  - la commande **[User-Agent]** donne l'identité du client ;
  - la commande **[Accept]** indique quels types de document sont acceptés par le client ;
  - la commande **[Accept-Language]** indique dans quelle langue sont souhaités les documents demandés s'ils existent en plusieurs langues ;
  - la commande **[Connection]** indique le mode de connexion souhaité : **[keep-alive]** indique que la connexion doit être maintenue jusqu'à ce que les échanges soient terminés ;
  - en **[7]** : le client termine ses commandes par une ligne vide ;

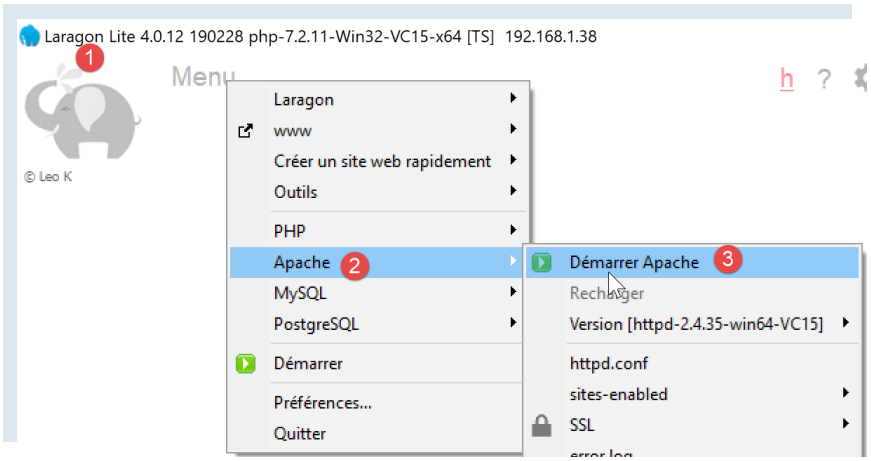
Nous terminons la connexion en terminant le serveur :

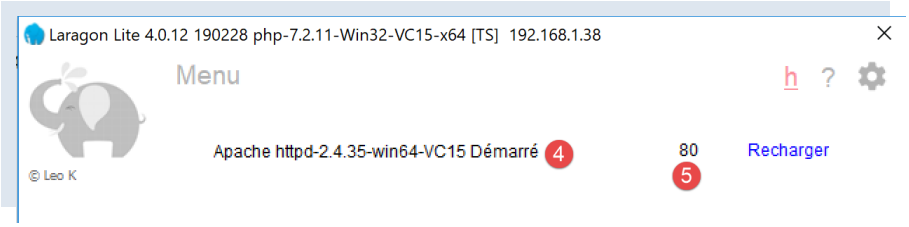
```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\u
server : Serveur générique lancé sur le port 0.0.0.0:1
server : Attente d'un client...
Commandes disponibles : [list, send id [texte], close
user : server : Client 1-DESKTOP-528I5CU-50509 connect
server : Attente d'un client...
client 1 : [GET / HTTP/1.1]
client 1 : [Host: localhost:100]
client 1 : [User-Agent: Mozilla/5.0 (Windows NT 10.0;
client 1 : [Accept: text/html,application/xhtml+xml,ap
client 1 : [Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.
client 1 : [Accept-Encoding: gzip, deflate]
client 1 : [DNT: 1]
client 1 : [Connection: keep-alive]
client 1 : [Upgrade-Insecure-Requests: 1]
client 1 : []
server : Client 2-DESKTOP-528I5CU-50510 connecté...
server : Attente d'un client...
quit8
server : fin du service9
```

1.16.4.2 Exemple 2

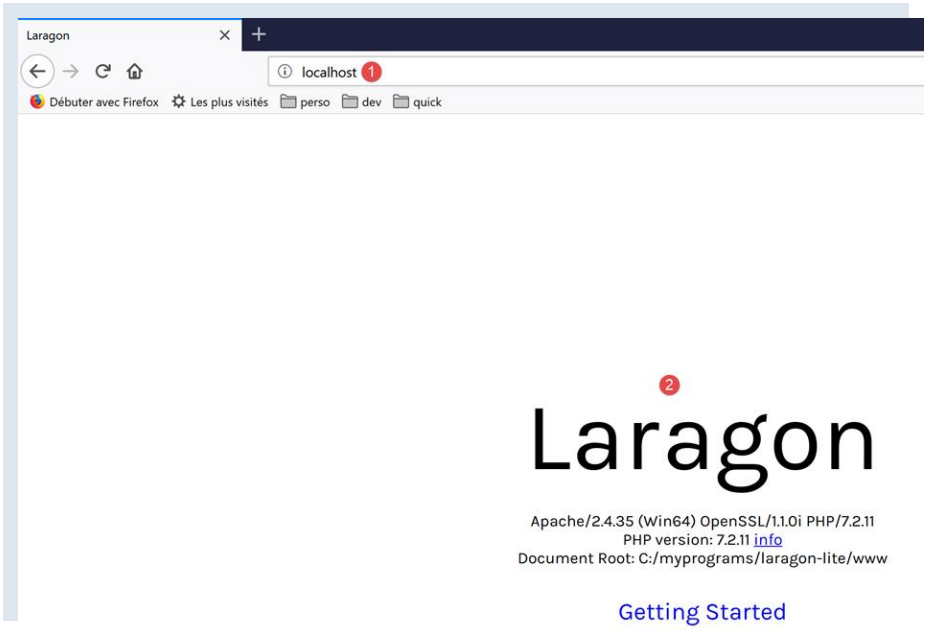
Maintenant que nous connaissons les commandes envoyées par un navigateur pour réclamer une URL, nous allons réclamer cette URL avec notre client TCP **[RawTcpClient]**. Le serveur Apache de Laragon sera notre serveur web.

Lançons Laragon puis le serveur web Apache :



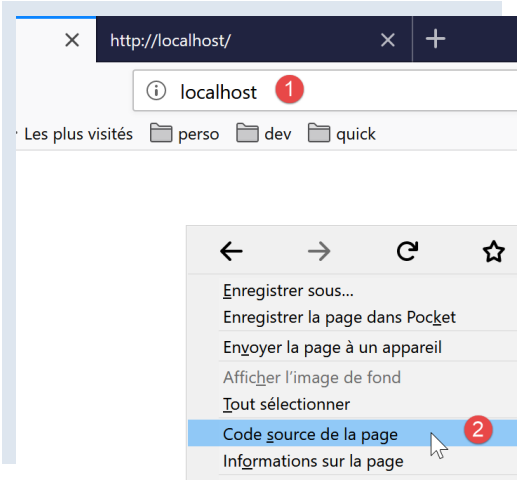


Maintenant avec un navigateur, demandons l'URL **[http://localhost:80]**. Ici nous ne précisons que le serveur **[localhost:80]** et pas d'URL de document. Dans ce cas c'est l'URL / qui est demandée, ç-à-d la racine du serveur web :



- en **[1]**, l'URL demandée. On a tapé initialement **[http://localhost:80]** et le navigateur (Firefox ici) l'a transformée simplement en **[localhost]** car le protocole **[http]** est implicite lorsqu'aucun protocole n'est mentionné et le port **[80]** est implicite lorsque le port n'est pas précisé ;
- en **[2]**, la page racine / du serveur web interrogé ;

Maintenant, visualisons le texte reçu par le navigateur :



- on clique droit sur la page reçue et on choisit l'option **[2]**. on obtient le code source suivant :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Laragon</title>

    <link
      href="https://fonts.googleapis.com/css?family=Karla:400"
      rel="stylesheet"
      type="text/css"
    />

    <style>
      HTML,
      body {
        height: 100%;
      }

      body {
        margin: 0;
        padding: 0;
        width: 100%;
        display: table;
        font-weight: 100;
        font-family: "Karla";
      }

      .container {
        text-align: center;
        display: table-cell;
        vertical-align: middle;
      }

      .content {
        text-align: center;
        display: inline-block;
      }

      .title {
        font-size: 96px;
      }

      .opt {
        margin-top: 30px;
      }

      .opt a {
        text-decoration: none;
        font-size: 150%;
      }

      a:hover {
```

```
52.         color: red;
53.       }
54.     </style>
55.   </head>
56.
57.   <body>
58.     <div class="container">
59.       <div class="content">
60.         <div class="title" title="Laragon">Laragon</div>
61.
62.         <div class="info">
63.           <br />
64.           Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11<br />
65.           PHP version: 7.2.11
66.           <span><a title="phpinfo()" href="/?q=info">info</a></span
67.           ><br />
68.           Document Root: C:/myprograms/laragon-lite/www<br />
69.         </div>
70.         <div class="opt">
71.           <div>
72.             <a title="Getting Started" href="https://laragon.org/docs"
73.               >Getting Started</a
74.             >
75.           </div>
76.         </div>
77.       </div>
78.     </div>
79.   </body>
80. </html>
```

Maintenant demandons l'URL **[http://localhost:80]** avec notre client TCP :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpClient localhost 80
Client [DESKTOP-528I5CU:50881] connecté au serveur [localhost-80] 2                1
Tapez vos commandes (quit pour arrêter) : 3
```

- en **[1]**, nous nous connectons au port 80 du serveur *localhost*. C'est là qu'opère le serveur web de Laragon ;

Nous tapons maintenant les commandes que nous avons découvertes dans le paragraphe précédent :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpClient localhost 80
Client [DESKTOP-528I5CU:50881] connecté au serveur [localhost-80]
Tapez vos commandes (quit pour arrêter) :
GET Perte de la connexion avec le serveur...

C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpClient localhost 80
Client [DESKTOP-528I5CU:50886] connecté au serveur [localhost-80]
Tapez vos commandes (quit pour arrêter) :
GET / HTTP/1.1 1
Host: localhost:80 2
3
<-- [HTTP/1.1 200 OK] 4
<-- [Date: Thu, 16 May 2019 14:24:39 GMT]
<-- [Server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11]
<-- [X-Powered-By: PHP/7.2.11]
<-- [Content-Length: 1781]
<-- [Content-Type: text/html; charset=UTF-8]
<-- [] 5
<-- [<!DOCTYPE html>] 6
<-- [<html>]
<-- [    <head>]
<-- [        <title>Laragon</title>]
<-- []
```

- en **[1]**, la commande **[GET]**. On demande la racine / du serveur web ;
- en **[2]**, la commande **[Host]** ;
- ce sont les deux seules commandes indispensables. Pour les autres commandes, le serveur web prendra des valeurs par défaut ;
- en **[3]**, la ligne vide qui doit terminer les commandes du client ;
- dessous la ligne 3, vient la réponse du serveur web ;
- en **[4]** jusqu'à la ligne vide **[5]** viennent les entêtes HTTP de la réponse du serveur ;
- après la ligne **[5]** vient le document HTML demandé **[6]** ;

Nous tapons **[quit]** pour terminer le client et nous chargeons le fichier de logs **[localhost-80.txt]** :

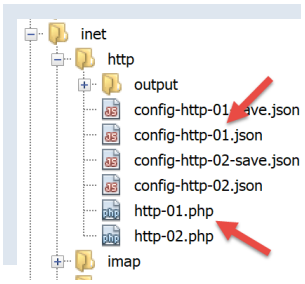
```
1   --> [GET / HTTP/1.1]
2   --> [Host: localhost:80]
3   --> []
4   <-- [HTTP/1.1 200 OK]
5   <-- [Date: Thu, 16 May 2019 14:24:39 GMT]
6   <-- [Server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11]
7   <-- [X-Powered-By: PHP/7.2.11]
8   <-- [Content-Length: 1781]
9   <-- [Content-Type: text/HTML; charset=UTF-8]
10  <-- []
11  <-- [<!DOCTYPE HTML>]
12  <-- [<HTML>]
13  <-- [    <head>]
14  <-- [        <title>Laragon</title>]
15  <-- []
16  <-- [        <link href="https://fonts.googleapis.com/css?family=Karla:400" rel="stylesheet"
    type="text/css">]
17  <-- []
18  <-- [        <style>]
19  <-- [            HTML, body {]
20  <-- [                height: 100%;]
21  <-- [            }]
22  <-- []
23  <-- [            body {]
24  <-- [                margin: 0;]
25  <-- [                padding: 0;]
26  <-- [                width: 100%;]
27  <-- [                display: table;]
28  <-- [                font-weight: 100;]
29  <-- [                font-family: 'Karla';]
30  <-- [            }]
31  <-- []
32  <-- [            .container {]
33  <-- [                text-align: center;]
34  <-- [                display: table-cell;]
35  <-- [                vertical-align: middle;]
36  <-- [            }]
37  <-- []
38  <-- [            .content {]
39  <-- [                text-align: center;]
40  <-- [                display: inline-block;]
41  <-- [            }]
42  <-- []
43  <-- [            .title {]
44  <-- [                font-size: 96px;]
45  <-- [            }]
46  <-- []
47  <-- [            .opt {]
48  <-- [                margin-top: 30px;]
49  <-- [            }]
50  <-- []
51  <-- [            .opt a {]
52  <-- [              text-decoration: none;]
53  <-- [              font-size: 150%;]
54  <-- [            }]
55  <-- [            ]
56  <-- [            a:hover {]
57  <-- [              color: red;]
58  <-- [            }]
59  <-- [        </style>]
60  <-- [    </head>]
61  <-- [    <body>]
62  <-- [        <div class="container">]
63  <-- [            <div class="content">]
64  <-- [                <div class="title" title="Laragon">Laragon</div>]
65  <-- [     ]
66  <-- [                <div class="info"><br />]
67  <-- [                      Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11<br />]
68  <-- [                      PHP version: 7.2.11   <span><a title="phpinfo()"
    href="/?q=info">info</a></span><br />]
69  <-- [                      Document Root: C:/myprograms/laragon-lite/www<br />]
70  <-- []
71  <-- [                </div>]
72  <-- [                <div class="opt">]
```

```
73  <-- [                  <div><a title="Getting Started" href="https://laragon.org/docs">Getting
    Started</a></div>]
74  <-- [                </div>]
75  <-- [            </div>]
76  <-- []
77  <-- [        </div>]
78  <-- [    </body>]
79  <-- [</HTML>]
```

- lignes 11-79 : le document HTML reçu. Dans l'exemple précédent, Firefox avait reçu le même ;

Nous avons désormais les bases pour programmer un client TCP qui demanderait une URL.

1.16.4.3 Exemple 3



Le script **[http-01.php]** est un client HTTP configuré par le fichier jSON **[config-http-01.json]**. Le contenu de celui-ci est le suivant :

```
1.  {
2.    "localhost": {
3.      "port": 80,
4.      "GET": "/",
5.      "Host": "localhost:80",
6.      "User-Agent": "client PHP",
7.      "Accept": "text/HTML",
8.      "Accept-Language": "fr",
9.      "endOfLine": "\r\n"
10.   }
11. }
```

- ligne 2 : le nom de la machine hébergeant le serveur web à atteindre ;
- ligne 3 : le port sur lequel opère ce serveur web ;
- ligne 4 : l'URL du document désiré ;
- ligne 5 : la machine cible sous la forme machine:port ;
- ligne 6 : l'identification du client HTTP : on peut mettre ce qu'on veut ;
- ligne 7 : le type de document accepté par le client, ici du texte HTML ;
- ligne 8 : la langue souhaitée pour le document demandé ;
- ligne 9 : la marque de fin de ligne pour les commandes envoyées par le client : en effet elle peut différer selon que le serveur est sur une machine Unix (\n) ou Windows (\r\n) ;

Le script **[http-01.php]** est le suivant :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5   //
6   // gestion des erreurs
7   // error_reporting(E_ALL & E_STRICT);
8   // ini_set("display_errors", "on");
9   //
10  // constantes
11  const CONFIG_FILE_NAME = "config-http-01.json";
12  //
13  // on récupère la configuration
14  $config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true);
15  // otenir le texte HTML des URL du fichier de configuration
16  foreach ($config as $site => $protocole) {
```

```php
  // lecture page index du site $ite
  $résultat = getURL($site, $protocole);
  // affichage résultat
  print "$résultat\n";
}//for
// fin
exit;

//--------------------------------------------------------------------
function getURL(string $site, array $protocole, $suivi = TRUE): string {
  // lit l'URL $site["GET"] et la stocke dans le fichier $site.HTML
  // le dialogue client /serveur se fait selon le protocole $protocole
  //
  // ouverture d'une connexion sur le port de $site
  $erreurNumber = 0;
  $erreur = "";
  $connexion = fsockopen($site, $protocole["port"], $erreurNumber, $erreur);
  // retour si erreur
  if ($connexion === FALSE) {
    return "Echec de la connexion au site (" . $site . " ," . $protocole["port"] . " : $erreur";
  }
  // $connexion représente un flux de communication bidirectionnel
  // entre le client (ce programme) et le serveur web contacté
  // ce canal est utilisé pour les échanges de commandes et d'informations
  // le protocole de dialogue est HTTP
  //
  // création du fichier $site.HTML
  $HTML = fopen("output/$site.HTML", "w");
  if ($HTML === FALSE) {
    // fermeture connexion client / serveur
    fclose($connexion);
    // retour erreur
    return "Erreur lors de la création du fichier $site.HTML";
  }
  // le client va commencer le dialogue HTTP avec le serveur
  if ($suivi) {
    print "Client : début de la communication avec le serveur [$site] ----------------------------\n";
  }
  // selon les serveurs, les lignes du client doivent se terminer par \n ou \r\n
  $endOfLine = $protocole["endOfLine"];
  // par simplification, on ne teste pas les cas d'erreur dans la communication client /serveur
  // le client envoie la commande GET pour demander l'URL $protocole["GET"]
  // syntaxe GET URL HTTP/1.1
  $commande = "GET " . $protocole["GET"] . " HTTP/1.1$endOfLine";
  // suivi ?
  if ($suivi) {
    print "--> $commande";
  }
  // on envoie la commande au serveur
  fputs($connexion, $commande);
  // émission des autres entêtes HTTP
  foreach ($protocole as $verb => $value) {
    if ($verb !== "GET" && $verb != "port"" && $verb !="endOfLine") {
      // on construit la commande
      $commande = "$verb: $value$endOfLine";
      // suivi ?
      if ($suivi) {
        print "--> $commande";
      }
      // on envoie la commande au serveur
      fputs($connexion, $commande);
    }
  }
  // les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
  fputs($connexion, $endOfLine);
  //
  // le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
  // ses données puis fermer le canal. Le client lit donc tout ce qui arrive de $connexion
  // jusqu'à la fermeture du canal
  //
  // on lit tout d'abord les entêtes HTTP envoyés par le serveur
  // ils se terminent eux-aussi par une ligne vide
  if ($suivi) {
    print "Réponse du serveur [$site] ----------------------------\n";
  }
  $fini = FALSE;
  while (!$fini && $ligne = fgets($connexion, 1000)) {
```

```
94      // a-t-on une ligne vide ?
95      $champs = [];
96      preg_match("/^(.*?)\s+$/", $ligne, $champs);
97      if ($champs[1] !== "") {
98        if ($suivi) {
99          // on affiche l'entête HTTP
100         print "<-- " . $champs[1] . "\n";
101       }
102     } else {
103       // c'était la ligne vide - les entêtes HTTP sont terminés
104       $fini = TRUE;
105     }
106   }
107   // on lit le document HTML qui va suivre la ligne vide
108   while ($ligne = fgets($connexion, 1000)) {
109     // on mémorise la ligne dans le fichier HTML du site
110     fputs($HTML, $ligne);
111   }
112   // le serveur a fermé la connexion -  le client la ferme à son tour
113   fclose($connexion);
114   // fermeture du fichier $HTML
115   fclose($HTML);
116   // retour
117   return "Fin de la communication avec le site [$site]. Vérifiez le fichier [$site.HTML]";
118 }
```

Commentaires du code :

- ligne 14 : le fichier de configuration est exploité pour créer un dictionnaire :
  - les clés du dictionnaire sont les serveurs web à interroger ;
  - les valeurs fixent le protocole HTTP à respecter ;
- lignes 16-21 : on boucle sur la liste des serveurs web de la configuration ;
- ligne 26 : la fonction *getURL($site,$protocole,$suivi)* demande un document du site web $*site* et le stocke dans le fichier texte $*site*.HTML.Par défaut, les échanges client/serveur sont logués sur la console ($suivi=TRUE) ;
- ligne 33 : la fonction *fsockopen($site,$port,$errNumber,$erreur)* permet de créer une connexion avec un service TCP / IP travaillant sur le port $*port* de la machine $*site*. Si la connexion échoue, **[$errNumber]** est un n° d'erreur et **[$erreur]** le message d'erreur associé. Une fois la connexion client / serveur ouverte, de nombreux services TCP / IP échangent des lignes de texte. C'est le cas ici du protocole HTTP (HyperText Transfer Protocol). Le flux du serveur parvenant au client peut alors être traité comme un fichier texte lu avec **[fgets]**. Il en est de même pour le flux partant du client vers le serveur qui peut être écrit avec **[fputs]** ;
- lignes 44-50 : création du fichier **[$site.HTML]** dans lequel on stockera le document HTML reçu ;
- ligne 60 : la première commande du client doit être la commande **[GET URL HTTP/1.1]** ;
- ligne 66 : la fonction *fputs* permet au client d'envoyer des données au serveur. Ici la ligne de texte envoyée a la signification suivante : "Je veux (GET) la page **[URL]** du site web auquel je suis connecté. Je travaille avec le protocole HTTP version 1.1" ;
- lignes 68-79 : on envoie les autres lignes du protocole HTTP **[Host, User-Agent, Accept, Accept-Language]**. Leur ordre n'importe pas ;
- ligne 81 : on envoie une ligne vide au serveur pour signifier que le client a terminé d'envoyer ses entêtes HTTP et qu'il attend désormais le document demandé ;
- lignes 92-106 : le serveur va tout d'abord envoyer une série d'entêtes HTTP qui vont donner diverses informations sur le document demandé. Ces entêtes se terminent par une ligne vide ;
- ligne 93 : on lit une ligne envoyée par le serveur avec la fonction PHP **[fgets]** ;
- ligne 96 : on récupère le corps de la ligne sans les espaces (blancs, marque de fin de ligne) de la fin de ligne ;
- ligne 97 : on regarde si on a récupéré la ligne vide qui marque la fin des entêtes HTTP envoyés par le serveur ;
- lignes 98-101 : si on est en mode **[suivi]**, l'entête HTTP reçu est affiché à la console ;
- lignes 108-111 : les lignes de texte de la réponse du serveur peuvent être lues ligne par ligne avec une boucle *while* et enregistrées dans le fichier texte **[output/$site.HTML]**. Lorsque le serveur web a envoyé la totalité de la page qu'on lui a demandée, il ferme sa connexion avec le client. Côté client, cela sera détecté comme une fin de fichier ;

Résultats :

La console affiche les logs suivants :

```
1   Client : début de la communication avec le serveur [localhost] ----------------------------
2   --> GET / HTTP/1.1
3   --> Host: localhost:80
4   --> User-Agent: client PHP
5   --> Accept: text/HTML
6   --> Accept-Language: fr
7   Réponse du serveur [localhost] ----------------------------
8   <-- HTTP/1.1 200 OK
9   <-- Date: Thu, 16 May 2019 15:43:18 GMT
```

```
<-- Server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
<-- X-Powered-By: PHP/7.2.11
<-- Content-Length: 1781
<-- Content-Type: text/HTML; charset=UTF-8
Fin de la communication avec le site [localhost]. Vérifiez le fichier [localhost.HTML]
```

Dans notre exemple, le fichier **[output/localhost.html]** reçu est le suivant :

```
<!DOCTYPE HTML>
<HTML>
    <head>
        <title>Laragon</title>

        <link href="https://fonts.googleapis.com/css?family=Karla:400" rel="stylesheet" type="text/css">

        <style>
            HTML, body {
                height: 100%;
            }

            body {
                margin: 0;
                padding: 0;
                width: 100%;
                display: table;
                font-weight: 100;
                font-family: 'Karla';
            }

            .container {
                text-align: center;
                display: table-cell;
                vertical-align: middle;
            }

            .content {
                text-align: center;
                display: inline-block;
            }

            .title {
                font-size: 96px;
            }

            .opt {
                margin-top: 30px;
            }

            .opt a {
              text-decoration: none;
              font-size: 150%;
            }

            a:hover {
              color: red;
            }
        </style>
    </head>
    <body>
        <div class="container">
            <div class="content">
                <div class="title" title="Laragon">Laragon</div>

                <div class="info"><br />
                      Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11<br />
                      PHP version: 7.2.11   <span><a title="phpinfo()" href="/?q=info">info</a></span><br
/>
                      Document Root: C:/myprograms/laragon-lite/www<br />

                </div>
                <div class="opt">
                  <div><a title="Getting Started" href="https://laragon.org/docs">Getting Started</a></div>
                </div>
            </div>

        </div>
    </body>
```

```
69  </HTML>
```

Nous avons bien obtenu le même document qu'avec le navigateur Firefox.

1.16.4.4 Exemple 4

Dans cet exemple, nous allons montrer que le client HTTP que nous avons écrit est insuffisant. Faison évoluer le fichier de configuration **[config-http-01.json]** de la façon suivante :

```
1.  {
2.    "tahe.developpez.com": {
3.      "port": 443,
4.      "GET": "/",
5.      "Host": "sergetahe.com:443",
6.      "User-Agent": "script PHP 7",
7.      "Accept": "text/HTML",
8.      "Accept-Language": "fr",
9.      "endOfLine": "\n"
10.   }
11. }
```

Ici, nous allons demander l'URL **[http://tahe.developpez.com:443/]**. Le port 443 de la machine **[tahe.developpez.com]** est un port utilisé pour le protocole http sécurisé appelé https. Dans ce protocole, le dialogue client / serveur commence par un échange d'informations qui vont sécuriser la liaison. Le client doit alors parler le protocole **[HTTPS]** et non le protocole **[HTTP]**, ce que ne fait pas notre client.

Avec ce fichier de configuration, les résultats de la console sont les suivants :

```
1   Client : début de la communication avec le serveur [tahe.developpez.com] ----------------------------
2   --> GET / HTTP/1.1
3   --> Host: sergetahe.com:443
4   --> User-Agent: script PHP 7
5   --> Accept: text/HTML
6   --> Accept-Language: fr
7   Réponse du serveur [tahe.developpez.com] ----------------------------
8   <-- HTTP/1.1 400 Bad Request
9   <-- Date: Fri, 17 May 2019 13:02:26 GMT
10  <-- Server: Apache/2.4.25 (Debian)
11  <-- Content-Length: 454
12  <-- Connection: close
13  <-- Content-Type: text/HTML; charset=iso-8859-1
14  Fin de la communication avec le site [tahe.developpez.com]. Vérifiez le fichier
    [output/tahe.developpez.com.HTML]
```

- ligne 8 : le serveur **[tahe.developpez.com]** a répondu que la requête du client était incorrecte ;

Le contenu du fichier **[output/tahe.developpez.com.html]** est alors le suivant :

```
1   <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
2   <HTML><head>
3   <title>400 Bad Request</title>
4   </head><body>
5   <h1>Bad Request</h1>
6   <p>Your browser sent a request that this server could not understand.<br />
7   Reason: You're speaking plain HTTP to an SSL-enabled server port.<br />
8    Instead use the HTTPS scheme to access this URL, please.<br />
9   </p>
10  <hr>
11  <address>Apache/2.4.25 (Debian) Server at 2eurocents.developpez.com Port 443</address>
12  </body></HTML>
```

Le serveur dit clairement que nous n'avons pas utilisé le bon protocole.

Utilisons maintenant, le fichier de configuration suivant :

```
1.  {
2.    "sergetahe.com": {
3.      "port": 80,
4.      "GET": "/cours-tutoriels-de-programmation/",
5.      "Host": "sergetahe.com:80",
6.      "User-Agent": "script PHP 7",
7.      "Accept": "text/HTML",
8.      "Accept-Language": "fr",
```

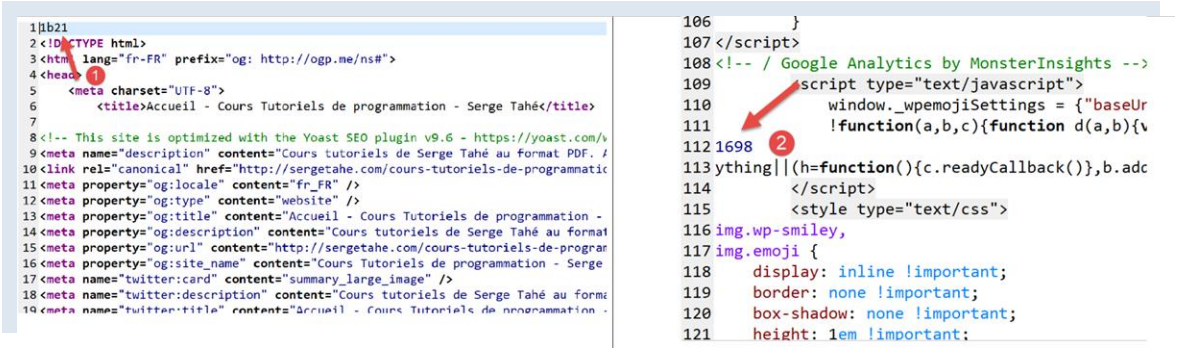
```
9.      "endOfLine": "\n"
10.   }
11. }
```

Les résultats console sont alors les suivants :

```
1   Client : début de la communication avec le serveur [sergetahe.com] ----------------------------
2   --> GET /cours-tutoriels-de-programmation/ HTTP/1.1
3   --> Host: sergetahe.com:80
4   --> User-Agent: script PHP 7
5   --> Accept: text/HTML
6   --> Accept-Language: fr
7   Réponse du serveur [sergetahe.com] ----------------------------
8   <-- HTTP/1.1 200 OK
9   <-- Date: Fri, 17 May 2019 13:36:06 GMT
10  <-- Content-Type: text/HTML; charset=UTF-8
11  <-- Transfer-Encoding: chunked
12  <-- Server: Apache
13  <-- X-Powered-By: PHP/7.0
14  <-- Vary: Accept-Encoding
15  <-- Set-Cookie: SERVERID68971=2621207|XN64y|XN64y; path=/
16  <-- Cache-control: private
17  <-- X-IPLB-Instance: 17106
18  Fin de la communication avec le site [sergetahe.com]. Vérifiez le fichier [output/sergetahe.com.HTML]
```

- la ligne 11 indique que le serveur envoie le document par morceaux ;

Cela se traduit par la présence de nombres dans le flux envoyé au client : chaque nombre indique au client le nombre de caractères du prochain morceau envoyé par le serveur. Voici ce que ça donne dans le fichier **[output/sergetahe.com.HTML]** :



- en **[1]** et **[2]**, la taille en hexadécimal des morceaux 1 et 2 du document ;

Un client HTTP correct ne devrait pas laisser ces nombres dans le document HTML final.

Voici un autre exemple :

```
1.  {
2.    "sergetahe.com": {
3.      "port": 80,
4.      "GET": "/cours-tutoriels-de-programmation",
5.      "Host": "sergetahe.com:80",
6.      "User-Agent": "script PHP 7",
7.      "Accept": "text/HTML",
8.      "Accept-Language": "fr",
9.      "endOfLine": "\n"
10.   }
11. }
```

Il ressemble à l'exemple précédent mais l'URL demandée en ligne 4 n'a pas le caractère / pour la terminer. Ce ne sont pas les mêmes URL. L'exécution du client HTTP donne alors les résultats console suivants :

```
1   Client : début de la communication avec le serveur [sergetahe.com] ----------------------------
2   --> GET /cours-tutoriels-de-programmation HTTP/1.1
3   --> Host: sergetahe.com:80
4   --> User-Agent: script PHP 7
5   --> Accept: text/HTML
```

```
6   --> Accept-Language: fr
7   Réponse du serveur [sergetahe.com] ----------------------------
8   <-- HTTP/1.1 301 Moved Permanently
9   <-- Date: Fri, 17 May 2019 13:47:00 GMT
10  <-- Content-Type: text/HTML; charset=iso-8859-1
11  <-- Content-Length: 262
12  <-- Server: Apache
13  <-- Location: http://sergetahe.com:80/cours-tutoriels-de-programmation/
14  <-- Set-Cookie: SERVERID68971=2621207|XN67V|XN67V; path=/
15  <-- Cache-control: private
16  <-- X-IPLB-Instance: 17095
17  Fin de la communication avec le site [sergetahe.com]. Vérifiez le fichier [output/sergetahe.com.HTML]
```

- la ligne 8 indique que le document demandé a changé d'URL. La nouvelle URL est donnée ligne 13. Notez cette fois-ci le caractère / qui termine la nouvelle URL ;

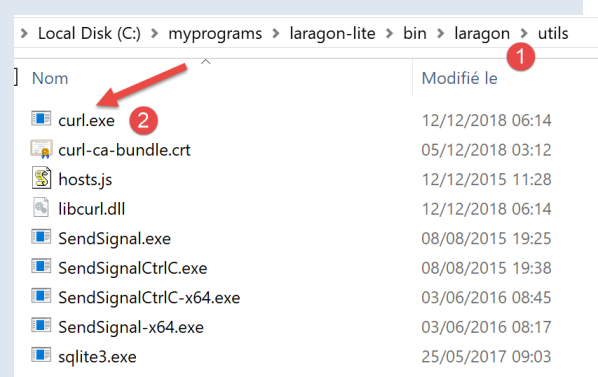
Le fichier **[output/serge.tahe.com.html]** est alors le suivant :

```
1   <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
2   <HTML><head>
3   <title>301 Moved Permanently</title>
4   </head><body>
5   <h1>Moved Permanently</h1>
6   <p>The document has moved <a href="http://sergetahe.com/cours-tutoriels-de-programmation/">here</a>.</p>
7   </body></HTML>
```

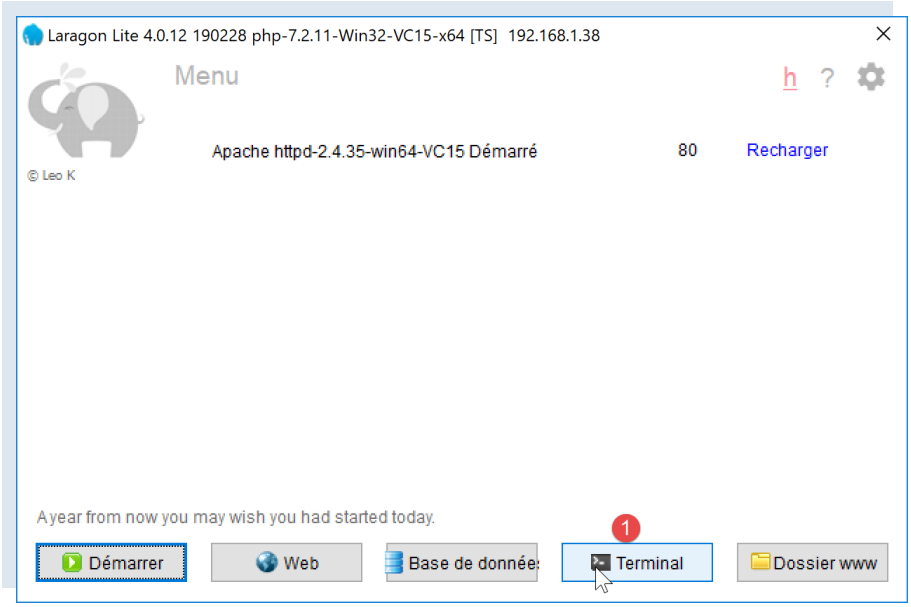
Un client HTTP devrait pouvoir suivre les redirections. Ici il devrait redemander automatiquement la nouvelle URL **[http://sergetahe.com/cours-tutoriels-de-programmation/]**.

1.16.4.5 Exemple 5

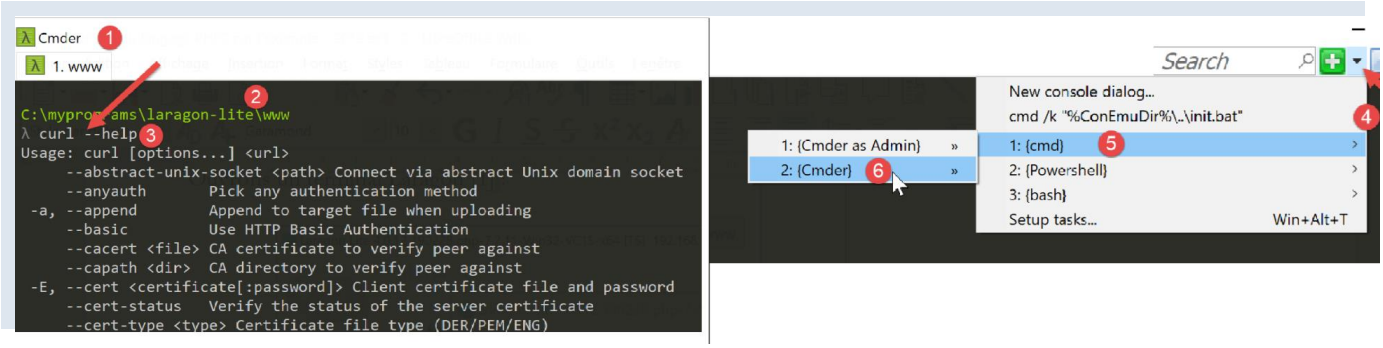
Les exemples précédents nous ont montré que notre client HTTP était insuffisant. Nous allons maintenant présenter un outil appelé **[curl]** qui permet de récupérer des documents web en gérant les difficultés mentionnées : protocole https, document envoyé par morceaux, redirections… L'outil **[curl]** a été installé avec Laragon :



Ouvrons un terminal Laragon **[1]** :



Dans le terminal nous tapons la commande suivante :



- en **[1]**, le type de la console ;
- en **[2]**, le dossier courant. Ce dossier est particulier : c'est là où le serveur Apache de Laragon vient chercher les documents qu'on lui demande. On évitera donc de polluer ce dossier ;
- en **[3]**, la commande tapée ;

Il est possible que la commande **[curl --help]** produise une erreur. La cause la plus probable est que vous n'avez pas le bon type de terminal. Dans ce cas, ouvrez un autre terminal avec les commandes **[4-6]** ;

La commande **[curl --help]** fait afficher toutes les options de configuration de **[curl]**. Il y en a plusieurs dizaines. Nous en utiliserons très peu. Pour demander une URL il suffit de taper la commande **[curl URL]**. Cette commande affichera sur la console le document demandé. Si on veut de plus les échanges HTTP entre le client et le serveur on écrira **[curl --verbose URL]**. Enfin pour enregistrer le document HTML demandé dans un fichier on écrira **[curl --verbose --output fichier URL]**.

Pour éviter de polluer le dossier **[www]** de Laragon, déplaçons-nous à un autre endroit du système de fichiers :

- en **[1]**, on se déplace dans le dossier **[c:\temp]**. Si ce dossier n'existe pas, vous pouvez le créer ou en choisir un autre ;
- en **[2]**, on crée un dossier appelé **[curl]** ;
- en **[3]**, on se positionne dessus ;
- en **[4]**, on liste son contenu. Il est vide ;

Assurez-vous que le serveur Apache de Laragon est lancé et avec **[curl]** demandez l'URL **[http://localhost/]** avec la commande **[curl -verbose -output localhost.html http://localhost/]**. On obtient les résultats suivants :

```
1   c:\Temp\curl
2   λ curl --verbose --output localhost.html http://localhost/
3     % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
4                                    Dload  Upload   Total   Spent    Left  Speed
5     0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   Trying ::1…
6   * TCP_NODELAY set
7   * Connected to localhost (::1) port 80 (#0)
8   > GET / HTTP/1.1
9   > Host: localhost
10  > User-Agent: curl/7.63.0
11  > Accept: */*
12  >
13  < HTTP/1.1 200 OK
14  < Date: Fri, 17 May 2019 14:32:47 GMT
15  < Server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
16  < X-Powered-By: PHP/7.2.11
17  < Content-Length: 1781
18  < Content-Type: text/HTML; charset=UTF-8
19  <
20  { [1781 bytes data]
21  100  1781  100  1781    0     0  14248      0 --:--:-- --:--:-- --:--:-- 14248
22  * Connection #0 to host localhost left intact
```

- lignes 8-12 : lignes envoyées par **[curl]** au serveur **[localhost]**. On reconnaît le protocole HTTP ;
- lignes 13-19 : lignes envoyées en réponse par le serveur ;
- ligne 13 : indique qu'on a bien eu le document demandé ;

Le fichier **[localhost.html]** contient le document demandé. Vous pouvez le vérifier en chargeant le fichier dans un éditeur de texte.

Maintenant demandons l'URL **[https://tahe.developpez.com:443/]**. Pour avoir cette URL, le client HTTP doit savoir parler HTTPS. C'est le cas du client **[curl]**.

Les résultats console sont les suivants :

```
1   c:\Temp\curl
```

```
2   λ curl --verbose --output tahe.developpez.com.HTML https://tahe.developpez.com:443/
3     % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
4                                    Dload  Upload   Total   Spent    Left  Speed
5     0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   Trying 87.98.130.52…
6   * TCP_NODELAY set
7   * Connected to tahe.developpez.com (87.98.130.52) port 443 (#0)
8   * ALPN, offering h2
9   * ALPN, offering http/1.1
10  * successfully set certificate verify locations:
11  *   CAfile: C:\myprograms\laragon-lite\bin\laragon\utils\curl-ca-bundle.crt
12    CApath: none
13  } [5 bytes data]
14  * TLSv1.3 (OUT), TLS handshake, Client hello (1):
15  } [512 bytes data]
16  * TLSv1.3 (IN), TLS handshake, Server hello (2):
17  { [108 bytes data]
18  * TLSv1.2 (IN), TLS handshake, Certificate (11):
19  { [2558 bytes data]
20  * TLSv1.2 (IN), TLS handshake, Server key exchange (12):
21  { [333 bytes data]
22  * TLSv1.2 (IN), TLS handshake, Server finished (14):
23  { [4 bytes data]
24  * TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
25  } [70 bytes data]
26  * TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
27  } [1 bytes data]
28  * TLSv1.2 (OUT), TLS handshake, Finished (20):
29  } [16 bytes data]
30  * TLSv1.2 (IN), TLS handshake, Finished (20):
31  { [16 bytes data]
32  * SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
33  * ALPN, server accepted to use http/1.1
34  * Server certificate:
35  *  subject: CN=*.developpez.com
36  *  start date: Apr  4 08:25:09 2019 GMT
37  *  expire date: Jul  3 08:25:09 2019 GMT
38  *  subjectAltName: host "tahe.developpez.com" matched cert's "*.developpez.com"
39  *  issuer: C=US; O=Let's Encrypt; CN=Let's Encrypt Authority X3
40  *  SSL certificate verify ok.
41  } [5 bytes data]
42  > GET / HTTP/1.1
43  > Host: tahe.developpez.com
44  > User-Agent: curl/7.63.0
45  > Accept: */*
46  >
47  { [5 bytes data]
48  < HTTP/1.1 200 OK
49  < Date: Fri, 17 May 2019 14:39:41 GMT
50  < Server: Apache/2.4.25 (Debian)
51  < X-Powered-By: PHP/5.3.29
52  < Vary: Accept-Encoding
53  < Transfer-Encoding: chunked
54  < Content-Type: text/HTML
55  <
56  { [6 bytes data]
57  100 96559    0 96559    0     0   163k      0 --:--:-- --:--:-- --:--:--  163k
58  * Connection #0 to host tahe.developpez.com left intact
```

- lignes 10-40 : les échanges client / serveur pour sécuriser la connexion : celle-ci sera chiffrée ;
- lignes 42-45 : les entêtes HTTP envoyés par le client **[curl]** au serveur ;
- ligne 48 : le document demandé a bien été trouvé ;
- ligne 53 : le document est envoyé par morceaux ;

[curl] gère correctement à la fois le protocole sécurisé HTTPS et le fait que le document soit envoyé par morceaux. Le document envoyé sera trouvé ici dans le fichier **[tahe.developpez.com.html]**.

Demandons maintenant l'URL **[http://sergetahe.com/cours-tutoriels-de-programmation]**. Nous avions vu que pour cette URL, il y avait une redirection vers l'URL **[http://sergetahe.com/cours-tutoriels-de-programmation/]** (avec un / à la fin).

Les résultats console sont alors les suivants :

```
1   c:\Temp\curl
2   λ curl --verbose --output sergetahe.com.HTML --location http://sergetahe.com/cours-tutoriels-de-
    programmation
```

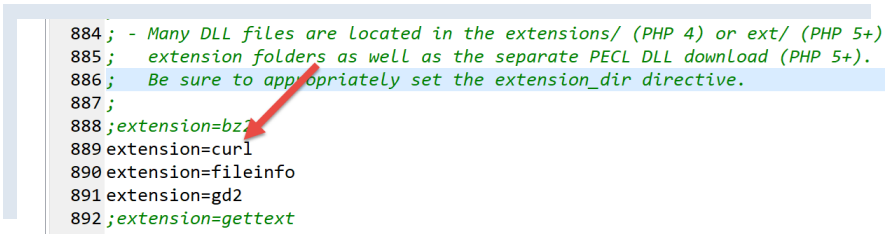
```
3     % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
4                                    Dload  Upload   Total   Spent    Left  Speed
5     0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   Trying 87.98.154.146…
6   * TCP_NODELAY set
7   * Connected to sergetahe.com (87.98.154.146) port 80 (#0)
8   > GET /cours-tutoriels-de-programmation HTTP/1.1
9   > Host: sergetahe.com
10  > User-Agent: curl/7.63.0
11  > Accept: */*
12  >
13  < HTTP/1.1 301 Moved Permanently
14  < Date: Fri, 17 May 2019 15:13:03 GMT
15  < Content-Type: text/HTML; charset=iso-8859-1
16  < Content-Length: 262
17  < Server: Apache
18  < Location: http://sergetahe.com/cours-tutoriels-de-programmation/
19  < Set-Cookie: SERVERID68971=2621207|XN7Pg|XN7Pg; path=/
20  < Cache-control: private
21  < X-IPLB-Instance: 17095
22  <
23  * Ignoring the response-body
24  { [262 bytes data]
25  100   262  100   262    0     0   1401      0 --:--:-- --:--:-- --:--:--  1401
26  * Connection #0 to host sergetahe.com left intact
27  * Issue another request to this URL: 'http://sergetahe.com/cours-tutoriels-de-programmation/'
28  * Found bundle for host sergetahe.com: 0x1c88548 [can pipeline]
29  * Could pipeline, but not asked to!
30  * Re-using existing connection! (#0) with host sergetahe.com
31  * Connected to sergetahe.com (87.98.154.146) port 80 (#0)
32    0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
33  > GET /cours-tutoriels-de-programmation/ HTTP/1.1
34  > Host: sergetahe.com
35  > User-Agent: curl/7.63.0
36  > Accept: */*
37  >
38  < HTTP/1.1 200 OK
39  < Date: Fri, 17 May 2019 15:13:04 GMT
40  < Content-Type: text/HTML; charset=UTF-8
41  < Transfer-Encoding: chunked
42  < Server: Apache
43  < X-Powered-By: PHP/7.0
44  < Vary: Accept-Encoding
45  < Set-Cookie: SERVERID68971=2621207|XN7Pg|XN7Pg; path=/
46  < Cache-control: private
47  < X-IPLB-Instance: 17095
48  <
49  { [14205 bytes data]
50  100 43101    0 43101    0     0  78795      0 --:--:-- --:--:-- --:--:--  168k
51  * Connection #0 to host sergetahe.com left intact
```

- ligne 2 : on utilise l'option **[--location]** pour indiquer qu'on veut suivre les redirections envoyées par le serveur ;
- ligne 13 : le serveur indique que le document demandé a changé d'URL ;
- ligne 18 : il indique la nouvelle URL du document demandé ;
- ligne 27 : **[curl]** émet une nouvelle requête vers cette fois la nouvelle URL ;
- ligne 33 : la nouvelle URL est utilisée ;
- ligne 38 : le serveur répond qu'il a trouvé le document demandé ;
- ligne 41 : il l'envoie par morceaux ;

Le document demandé sera trouvé dans le fichier **[sergetahe.com.HTML]**.

1.16.4.6 Exemple 6

PHP possède une extension appelée **[libcurl]** qui permet d'utiliser les capacités de l'outil **[curl]** dans un programme PHP. Il faut tout d'abord s'assurer que cette extension est activée dans le fichier **[php.ini]** décrit au paragraphe lien :



Assurez-vous que la ligne 889 ci-dessus est décommentée.

Nous allons écrire un script **[http-02.php]** qui exploitera le fichier de configuration jSON suivant :

```
1.  {
2.    "sergetahe.com": {
3.      "timeout": 5,
4.      "url": "http://sergetahe.com"
5.    },
6.    "tahe.developpez.com": {
7.      "timeout": 5,
8.      "url": "https://tahe.developpez.com"
9.    },
10.   "www.polytech-angers.fr": {
11.     "timeout": 5,
12.     "url": "http://www.polytech-angers.fr"
13.   },
14.   "localhost": {
15.     "timeout": 5,
16.     "url": "http://localhost"
17.   }
18. }
```

Chaque élément du dictionnaire **[clé, valeur]** a la structure suivante :

- *clé* : le nom d'un serveur web ;
- *valeur* est un dictionnaire avec les clés suivantes :
  - *timeout* : durée maximale d'attente de la réponse du serveur. Au-delà, le client se déconnectera ;
  - *url* : URL du document demandé ;

Le code du script **[http-02.php]** est le suivant :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5   //
6   // gestion des erreurs
7   //error_reporting(E_ALL & E_STRICT);
8   //ini_set("display_errors", "on");
9   //
10  // constantes
11  const CONFIG_FILE_NAME = "config-http-02.json";
12  //
13  // on récupère la configuration
14  $config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true);
15
16  // obtenir le texte HTML des URL du fichier de configuration
17  foreach ($config as $site => $infos) {
18    // lecture URL du site $ite
19    $résultat = getUrl($site, $infos["url"], $infos["timeout"]);
20    // affichage résultat
21    print "$résultat\n";
22  }//for
23  // fin
24  exit;
25
26  //-----------------------------------------------------------------------
27  function getUrl(string $site, string $url, int $timeout, $suivi = TRUE): string {
28    // lit l'URL $url et la stocke dans le fichier output/$site.HTML
29    //
```

```
30    // suivi
31    print "Client : début de la communication avec le serveur [$site] ----------------------------\n";
32
33    // Initialisation d'une session cURL
34    $curl = curl_init($url);
35    if ($curl === FALSE) {
36      // il y a eu une erreur
37      return "Erreur lors de l'initialisation de la session cURL pour le site [$site]";
38    }
39    // options de curl
40    $options = [
41      // mode verbose
42      CURLOPT_VERBOSE => true,
43      // nouvelle connexion - pas de cache
44      CURLOPT_FRESH_CONNECT => true,
45      // timeout de la requête (en secondes)
46      CURLOPT_TIMEOUT => $timeout,
47      CURLOPT_CONNECTTIMEOUT => $timeout,
48      // ne pas vérifier la validité des certificats SSL
49      CURLOPT_SSL_VERIFYPEER => false,
50      // suivre les redirections
51      CURLOPT_FOLLOWLOCATION => true,
52      // récupération du document demandé sous la forme d'une chaîne de caractères
53      CURLOPT_RETURNTRANSFER => true
54    ];
55
56    // paramétrage de curl
57    curl_setopt_array($curl, $options);
58    // Execution de la requête
59    $page_content = curl_exec($curl);
60    // Fermeture de la session cURL
61    curl_close($curl);
62
63    // exploitation du résultat
64    if ($page_content !== FALSE) {
65      // enregistrement du résultat dans $site.HTML
66      $result = file_put_contents("output/$site.HTML", $page_content);
67      if ($result === FALSE) {
68        // retour erreur
69        return "Erreur lors de la création du fichier [output/$site.HTML]";
70      }
71      // retour avec succès
72      return "Fin de la communication avec le serveur [$site]. Vérifiez le fichier [output/$site.HTML]";
73    } else {
74      // il y a eu une erreur de communication
75      return "Erreur de communication avec le serveur [$site]";
76    }
77  }
```

Commentaires

- ligne 14 : on exploite le fichier de configuration pour créer le dictionnaire **[$config]** ;
- lignes 17-22 : on boucle sur la liste de sites trouvés dans la configuration ;
- ligne 19 : pour chacun des sites, on appelle la fonction **[getUrl]** qui va télécharger l'URL **$infos[«url»]** avec un timeout **$infos[«timeout»]** ;
- ligne 34 : on démarre un session **[curl]**. **[curl_init]** ne fait pas encore de connexion au serveur web. Elle rend une ressource **[$curl]** qui va être un paramètre pour toutes les fonctions **[curl]** suivantes ;
- lignes 35-38 : si l'initialisation de la session **[curl]** échoue, la fonction **[curl_init]** rend le booléen FALSE ;
- lignes 40-54 : le dictionnaire **[$options]** va paramétrer la connexion **[curl]** au serveur ;
- ligne 57 : les options de la connexion sont transmises à la ressource **[$curl]** ;
- ligne 59 : connexion à l'URL demandée avec les options définies. A cause de l'option **[CURLOPT_RETURNTRANSFER => true]**, la fonction **[curl_exec]** rend comme résultat le document envoyé par le serveur comme une chaîne de caractères. La fonction **[curl_exec]** rend le booléen FALSE en cas d'échec de la connexion ;
- ligne 64 : on analyse le résultat de **[curl_exec]** ;
- ligne 66 : on enregistre la page reçue dans un fichier local ;
- lignes 69, 72, 75 : on rend le résultat de la fonction **[getUrl]** ;

Lorsqu'on exécute le script **[http-02.php]** on obtient les résultats console suivants :

```
1   * Rebuilt URL to: http://sergetahe.com/
2   Client : début de la communication avec le serveur [sergetahe.com] ----------------------------
3   *   Trying 87.98.154.146…
4   * TCP_NODELAY set
```

```
5   * Connected to sergetahe.com (87.98.154.146) port 80 (#0)
6   > GET / HTTP/1.1
7   Host: sergetahe.com
8   Accept: */*
9
10  < HTTP/1.1 302 Found
11  < Date: Sat, 18 May 2019 08:46:38 GMT
12  < Content-Type: text/HTML; charset=UTF-8
13  < Transfer-Encoding: chunked
14  < Server: Apache
15  < X-Powered-By: PHP/7.0
16  < Location: http://sergetahe.com/cours-tutoriels-de-programmation
17  < Set-Cookie: SERVERID68971=2621236|XN/Gc|XN/Gc; path=/
18  < X-IPLB-Instance: 17097
19  <
20  * Ignoring the response-body
21  * Connection #0 to host sergetahe.com left intact
22  * Issue another request to this URL: 'http://sergetahe.com/cours-tutoriels-de-programmation'
23  * Found bundle for host sergetahe.com: 0x1fee4ebe090 [can pipeline]
24  * Re-using existing connection! (#0) with host sergetahe.com
25  * Connected to sergetahe.com (87.98.154.146) port 80 (#0)
26  > GET /cours-tutoriels-de-programmation HTTP/1.1
27  Host: sergetahe.com
28  Accept: */*
29
30  < HTTP/1.1 301 Moved Permanently
31  < Date: Sat, 18 May 2019 08:46:38 GMT
32  < Content-Type: text/HTML; charset=iso-8859-1
33  < Content-Length: 262
34  < Server: Apache
35  < Location: http://sergetahe.com/cours-tutoriels-de-programmation/
36  < Set-Cookie: SERVERID68971=2621236|XN/Gc|XN/Gc; path=/
37  < Cache-control: private
38  < X-IPLB-Instance: 17097
39  <
40  * Ignoring the response-body
41  * Connection #0 to host sergetahe.com left intact
42  * Issue another request to this URL: 'http://sergetahe.com/cours-tutoriels-de-programmation/'
43  * Found bundle for host sergetahe.com: 0x1fee4ebe090 [can pipeline]
44  * Re-using existing connection! (#0) with host sergetahe.com
45  * Connected to sergetahe.com (87.98.154.146) port 80 (#0)
46  > GET /cours-tutoriels-de-programmation/ HTTP/1.1
47  Host: sergetahe.com
48  Accept: */*
49
50  < HTTP/1.1 200 OK
51  < Date: Sat, 18 May 2019 08:46:39 GMT
52  < Content-Type: text/HTML; charset=UTF-8
53  < Transfer-Encoding: chunked
54  < Server: Apache
55  < X-Powered-By: PHP/7.0
56  < Link: <http://sergetahe.com/cours-tutoriels-de-programmation/wp-json/>; rel="https://api.w.org/"
57  < Link: <http://sergetahe.com/cours-tutoriels-de-programmation/>; rel=shortlink
58  < Vary: Accept-Encoding
59  < Set-Cookie: SERVERID68971=2621236|XN/Gc|XN/Gc; path=/
60  < Cache-control: private
61  < X-IPLB-Instance: 17097
62  <
63  Fin de la communication avec le serveur [sergetahe.com]. Vérifiez le fichier [output/sergetahe.com.HTML]
64  Client : début de la communication avec le serveur [tahe.developpez.com] ----------------------------
65  * Connection #0 to host sergetahe.com left intact
66  * Rebuilt URL to: https://tahe.developpez.com/
67  *   Trying 87.98.130.52…
68  * TCP_NODELAY set
69  * Connected to tahe.developpez.com (87.98.130.52) port 443 (#0)
70  * ALPN, offering http/1.1
71  * successfully set certificate verify locations:
72  *   CAfile: C:\myprograms\laragon-lite\etc\ssl\cacert.pem
73    CApath: none
74  * SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
75  * ALPN, server accepted to use http/1.1
76  * Server certificate:
77  *  subject: CN=*.developpez.com
78  *  start date: Apr  4 08:25:09 2019 GMT
79  *  expire date: Jul  3 08:25:09 2019 GMT
80  *  subjectAltName: host "tahe.developpez.com" matched cert's "*.developpez.com"
81  *  issuer: C=US; O=Let's Encrypt; CN=Let's Encrypt Authority X3
```

```
*  SSL certificate verify ok.
> GET / HTTP/1.1
Host: tahe.developpez.com
Accept: */*

< HTTP/1.1 200 OK
< Date: Sat, 18 May 2019 08:46:42 GMT
< Server: Apache/2.4.25 (Debian)
< X-Powered-By: PHP/5.3.29
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/HTML
<
Fin de la communication avec le serveur [tahe.developpez.com]. Vérifiez le fichier [output/tahe.developpez.com.HTML]
Client : début de la communication avec le serveur [www.polytech-angers.fr] ----------
* Connection #0 to host tahe.developpez.com left intact
* Rebuilt URL to: http://www.polytech-angers.fr/
*   Trying 193.49.144.41…
* TCP_NODELAY set
* Connected to www.polytech-angers.fr (193.49.144.41) port 80 (#0)
> GET / HTTP/1.1
Host: www.polytech-angers.fr
Accept: */*

< HTTP/1.1 301 Moved Permanently
< Date: Sat, 18 May 2019 08:46:45 GMT
< Server: Apache/2.4.29 (Ubuntu)
< Location: http://www.polytech-angers.fr/fr/index.HTML
< Cache-Control: max-age=1
< Expires: Sat, 18 May 2019 08:46:46 GMT
< Content-Length: 339
< Content-Type: text/HTML; charset=iso-8859-1
<
* Ignoring the response-body
* Connection #0 to host www.polytech-angers.fr left intact
* Issue another request to this URL: 'http://www.polytech-angers.fr/fr/index.HTML'
* Found bundle for host www.polytech-angers.fr: 0x1fee4ebe390 [can pipeline]
* Re-using existing connection! (#0) with host www.polytech-angers.fr
* Connected to www.polytech-angers.fr (193.49.144.41) port 80 (#0)
> GET /fr/index.HTML HTTP/1.1
Host: www.polytech-angers.fr
Accept: */*

< HTTP/1.1 200
< Date: Sat, 18 May 2019 08:46:46 GMT
< Server: Apache/2.4.29 (Ubuntu)
< X-Cocoon-Version: 2.1.13-dev
< Accept-Ranges: bytes
< Last-Modified: Sat, 18 May 2019 08:01:36 GMT
< Content-Type: text/HTML; charset=UTF-8
< Content-Length: 47372
< Vary: Accept-Encoding
< Cache-Control: max-age=1
< Expires: Sat, 18 May 2019 08:46:47 GMT
< Content-Language: fr
<
* Connection #0 to host www.polytech-angers.fr left intact
Fin de la communication avec le serveur [www.polytech-angers.fr]. Vérifiez le fichier [output/www.polytech-angers.fr.HTML]
Client : début de la communication avec le serveur [localhost] ----------
* Rebuilt URL to: http://localhost/
*   Trying ::1…
* TCP_NODELAY set
* Connected to localhost (::1) port 80 (#0)
> GET / HTTP/1.1
Host: localhost
Accept: */*

< HTTP/1.1 200 OK
< Date: Sat, 18 May 2019 08:46:47 GMT
< Server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
< X-Powered-By: PHP/7.2.11
< Content-Length: 1781
< Content-Type: text/HTML; charset=UTF-8
<
* Connection #0 to host localhost left intact
```

```
157
158 Fin de la communication avec le serveur [localhost]. Vérifiez le fichier [output/localhost.HTML]
```

Commentaires

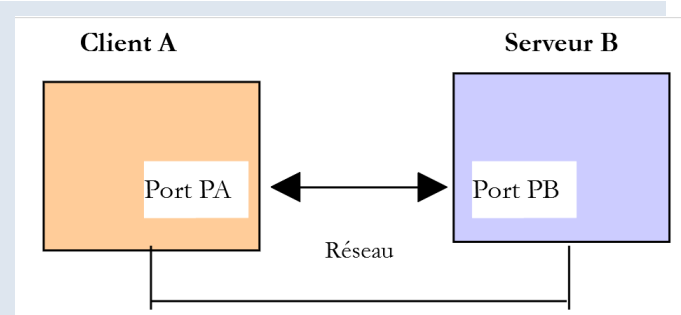
- on obtient les mêmes échanges qu'avec l'outil **[curl]** ;
- en vert, les logs du script ;
- en bleu, les commandes envoyées au serveur ;
- en jaune, les commandes reçues en réponse par le client ;

1.16.4.7 Conclusion

Nous avons, dans ce paragraphe, découvert le protocole HTTP et avons écrit un script **[http-02.php]** capable de télécharger une URL du web.

1.16.5 Le protocole SMTP (Simple Mail Transfer Protocol)

1.16.5.1 Introduction

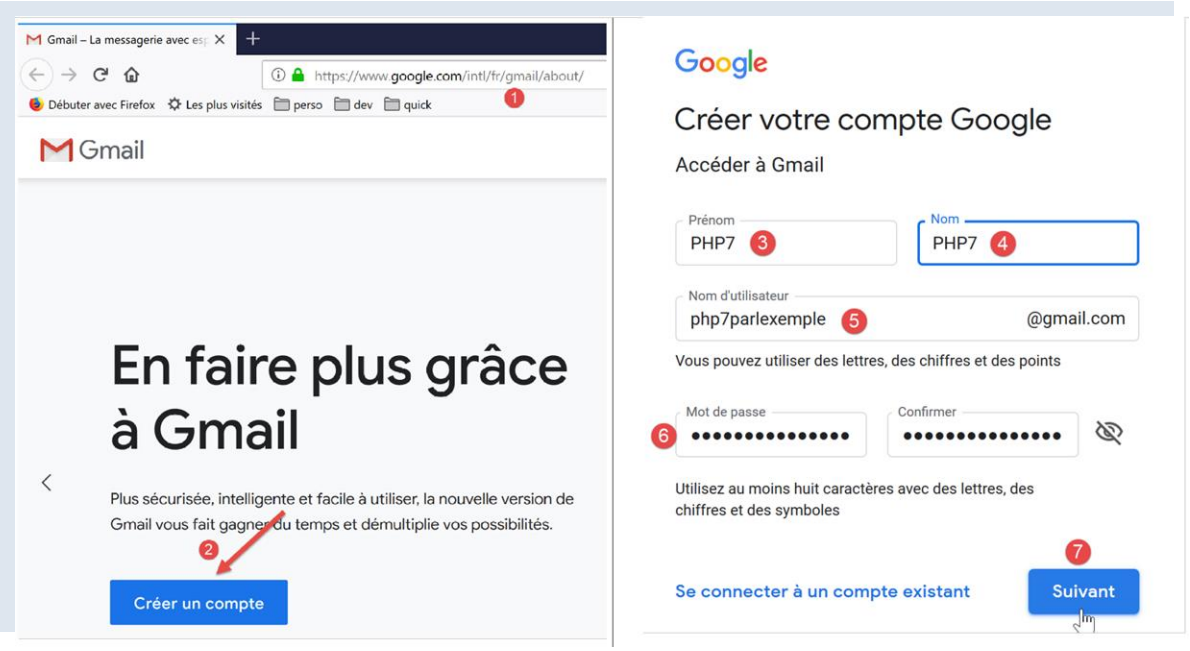


Dans ce chapitre :

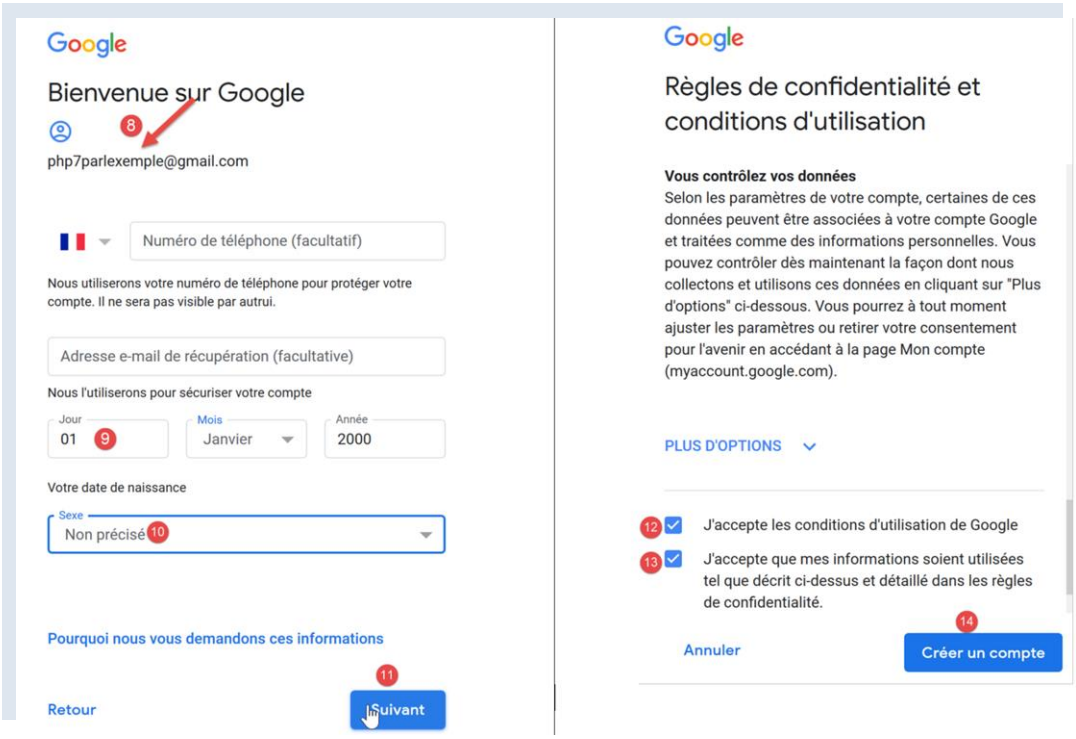
- **[Serveur B]** sera un serveur SMTP local que nous installerons ;
- **[Client A]** sera un client SMTP de diverses formes :
  - le client **[RawTcpClient]** pour découvrir le protocole SMTP ;
  - un script PHP rejouant le protocole SMTP du client **[RawTcpClient]** ;
  - un script PHP utilisant la bibliothèque **[SwiftMailServer]** permettant d'envoyer toutes sortes de mails ;

1.16.5.2 Création d'une adresse [gmail]

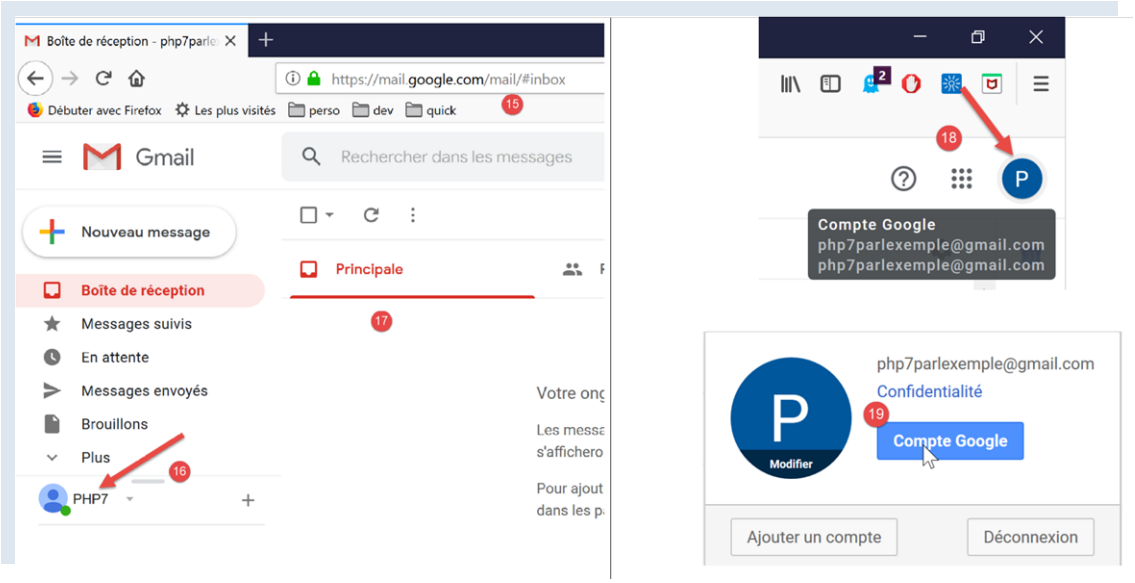
Pour faire nos tests SMTP, nous aurons besoin d'une adresse mail à qui écrire. Nous allons créer pour cela une adresse sur Gmail :



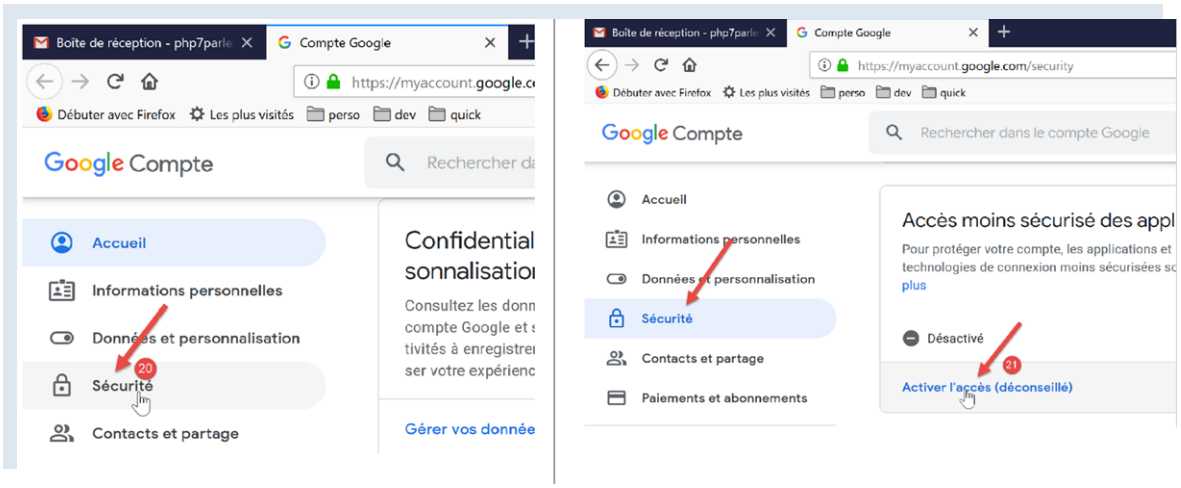
- en **[5]**, nous créons l'utilisateur **[php7parlexemple]** (choisissez autre chose) ;
- en **[6]**, le mot de passe sera lui **[PHP7parlexemple]** (choisissez autre chose) ;
- en **[7]**, nous validons ces informations ;



- remplir les cases **[9-10]** puis valider (11) ;
- accepter les conditions d'utilisation de Google (12-13) puis valider (14) ;



- en **[15]**, la boîte de réception (Inbox) de l'utilisateur **[PHP7]** (16) ;
- en **[17]**, cet utilisateur a une boîte de réception vide ;
- en **[18-19]**, connectez-vous au compte Google de l'utilisateur **[php7parlexemple@gmail.com]**. Nous allons configurer la sécurité du compte ;

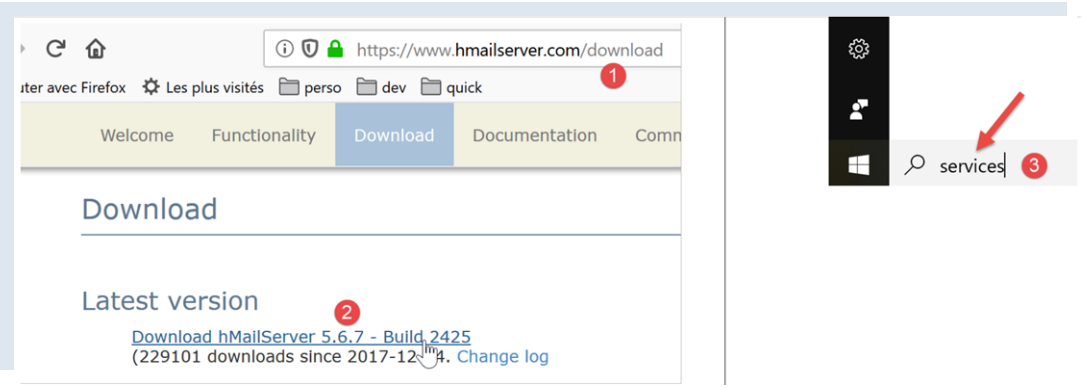


- en **[21]**, autorisez d'autres applications que celles de Google à exploiter le compte **[php7parlexemple]**. Si on ne fait pas ça, notre serveur local de mails **[hMailServer]** ne pourra pas communiquer avec le serveur SMTP de Gmail ;

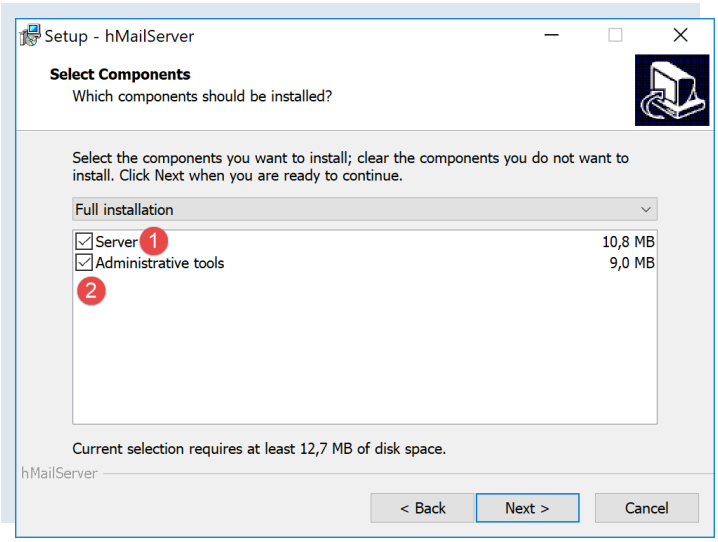
1.16.5.3 Installation d'un serveur SMTP

Pour nos tests, nous installerons le serveur de mail **[hMailServer]** qui est à la fois un serveur SMTP permettant d'envoyer des mails, un serveur POP3 (Post Office Protocol) permettant de lire les mails stockés sur le serveur, un serveur IMAP (Internet Message Access Protocol) qui lui aussi permet de lire les mails stockés sur le serveur mais va au-delà. Il permet notamment de gérer le stockage des mails sur le serveur.

Le serveur de mail **[hMailServer]** est disponible à l'URL **[https://www.hmailserver.com/]** (mai 2019).



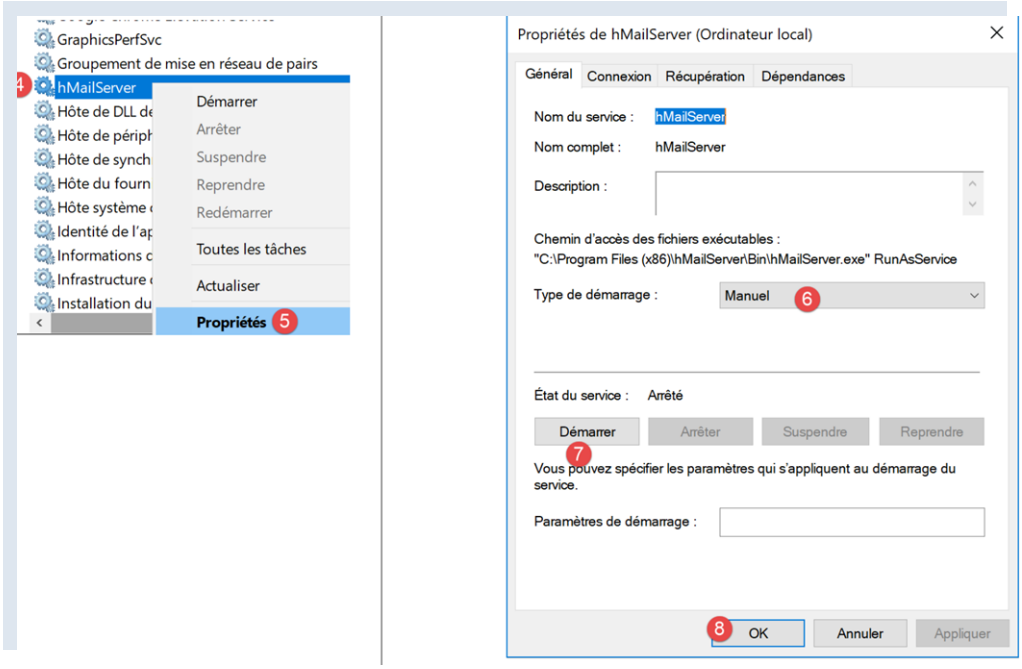
Au cours de l'installation, certains renseignements vous seront demandés :



- en **[1-2]**, sélectionnez à la fois le serveur de mails et les outils pour l'administrer ;
- durant l'installation le mot de l'administrateur vous sera demandé : **notez l**e, car il vous sera nécessaire ;

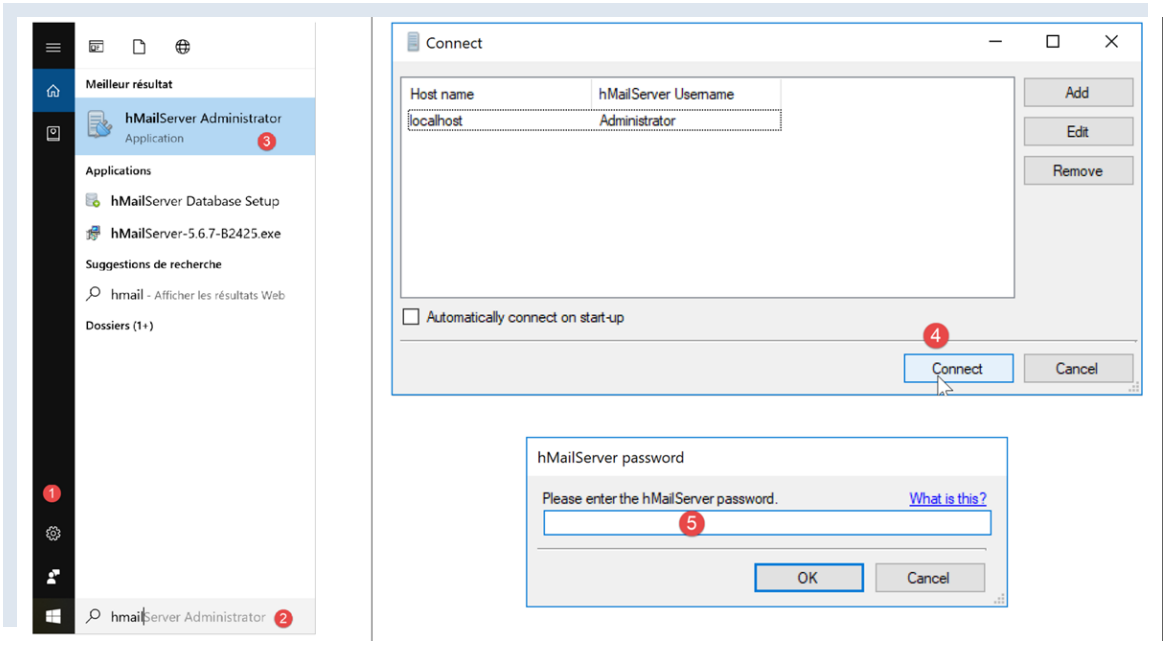
[hMailServer] s'installe comme un service Windows lancé automatiquement au démarrage de la machine. Il est préférable de choisir un démarrage manuel :

- en **[3]**, on tape **[services]** dans la zone de saisie de la barre d'état ;

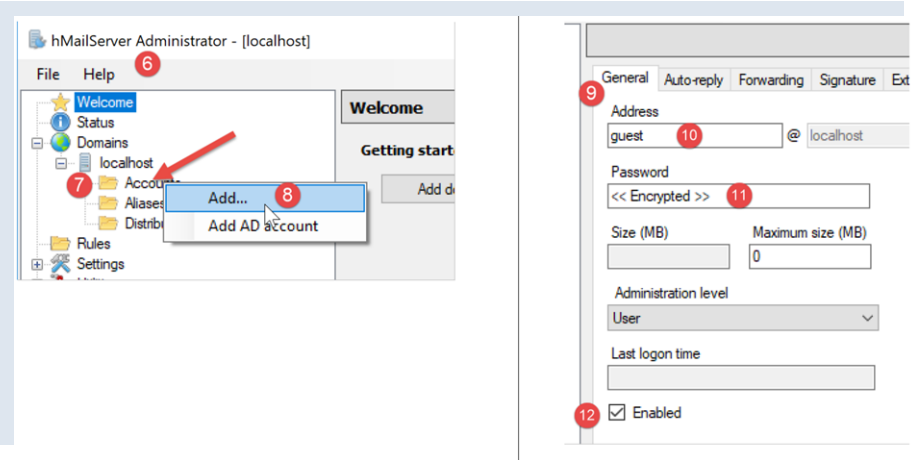


- en **[4-8]**, on met le service en mode **[manuel]** (6), on le lance (7) ;

Une fois démarré, le serveur **[hMailServer]** doit être configuré. Le serveur a été installé avec un programme d'administration **[hMailServer Administrator]** :

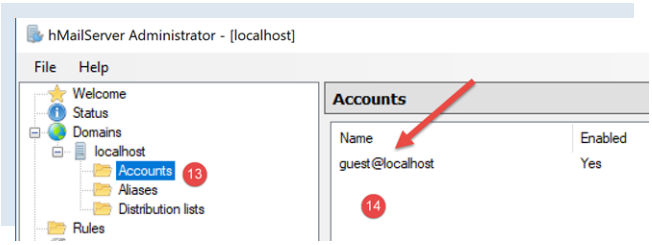


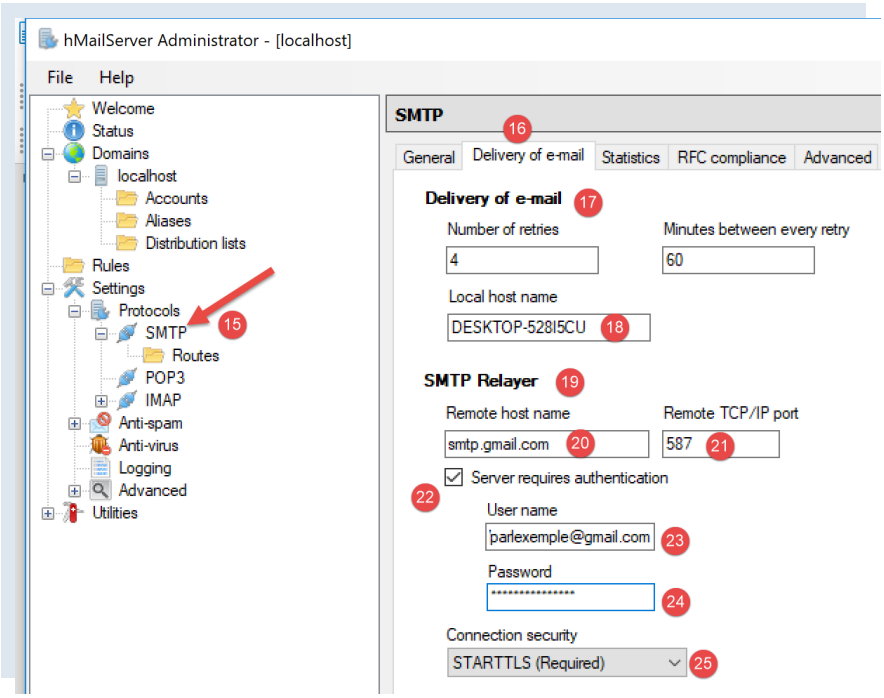
- en **[2]**, dans la zone de saisie de la barre d'état, taper **[hmailserver]** ;
- en **[3]**, lancer l'administrateur ;
- en **[4]**, connecter l'administrateur au serveur **[hMailServer]** ;
- en **[5]**, taper le mot de passe saisi lors de l'installation de **[hMailServer]** ;



Nous allons créer un compte utilisateur :

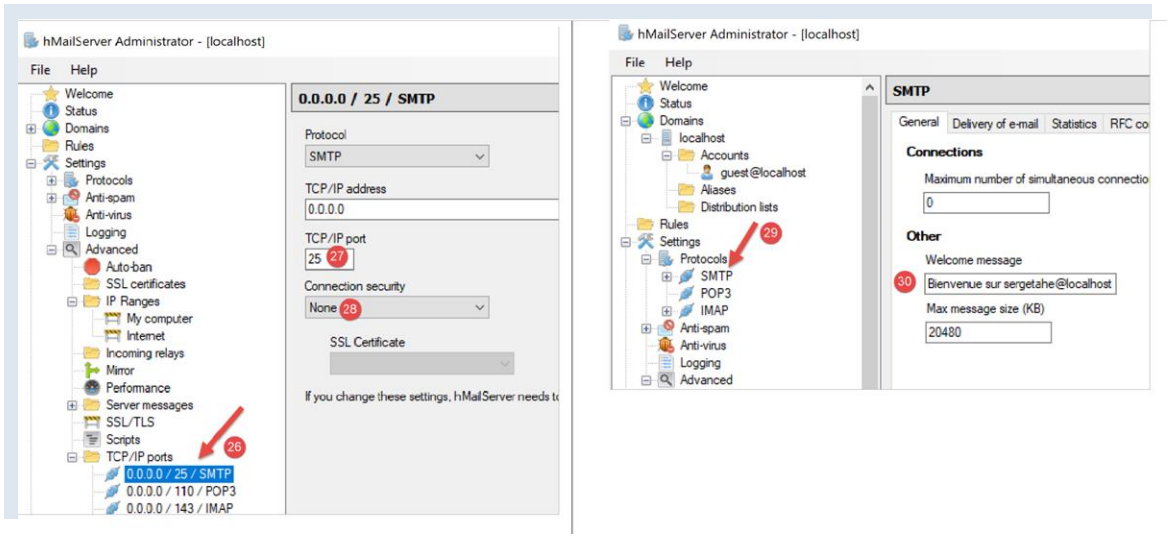
- cliquer droit sur **[Accounts]** (7) puis (8) pour ajouter un nouvel utilisateur ;
- dans l'onglet **[General]** (9), nous définissons un utilisateur **[guest]** (10) avec le mot de passe **[guest]** (11). Il aura l'adresse mail **[guest@localhost]** (10) ;
- en **[12]**, l'utilisateur **[guest]** est activé ;





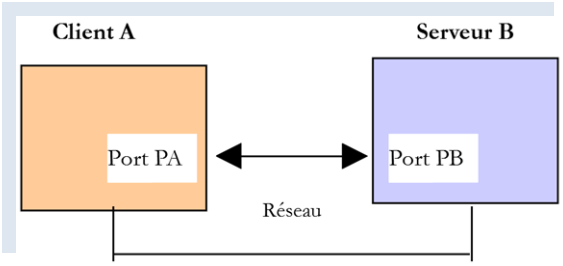
- en **[15]**, on configure le protocole SMTP du serveur de mail ;
- en **[16]**, on configure la distribution des mails ;
- en **[17]**, la configuration de la distribution des mails à destination de la machine hôte (localhost) ;
- en **[18]**, le nom de la machine locale (localhost). Le script du paragraphe lien vous permet d'avoir ce nom ;

- en **[19]**, on configure un serveur SMTP relais : il s'agit ici du serveur qui s'occupera de la distribution des mails non destinés à la machine locale (localhost) ;
- en **[20]**, le serveur SMTP de Gmail. Nous prenons Gmail car nous y avons créé un compte au paragraphe lien ;
- en **[21]**, le port SMTP de Gmail ;
- en **[22]**, le service SMTP de Gmail est un service sécurisé : il faut un compte Gmail pour y accéder ;
- en **[23]**, l'utilisateur **[php7parlexemple]** créé au paragraphe lien ;
- en **[24]**, le mot de passe de cet utilisateur : **[PHP7parlexemple]** créé au paragraphe lien ;
- en **[25]**, on indique le type de protocole de sécurité utilisé par Gmail ;



- en **[27]** le port du service SMTP ;
- en **[28]**, ce service ne nécessite pas d'authentification ;
- en **[30]**, mettez le message de bienvenue que le serveur SMTP enverra à ses clients ;

1.16.5.4 Le protocole SMTP



Nous allons découvrir le protocole SMTP avec l'environnement suivant :

- le client A sera le client TCP générique **[RawTcpClient]** ;
- le serveur B sera le serveur de mails **[hMailServer]** ;
- le client A demandera au serveur B de distribuer un courrier à l'utilisateur **[php7parlexemple@gmail.com]** ;
- nous vérifierons que cet utilisateur a bien reçu le mail envoyé ;

Nous lançons le client de la façon suivante :

```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpClient.exe --quit bye localhost 25
Client [DESKTOP-528I5CU:50387] connecté au serveur [localhost-25]
Tapez vos commandes (bye pour arrêter) :
<-- [220 Bienvenue sur sergetahe@localhost]
```

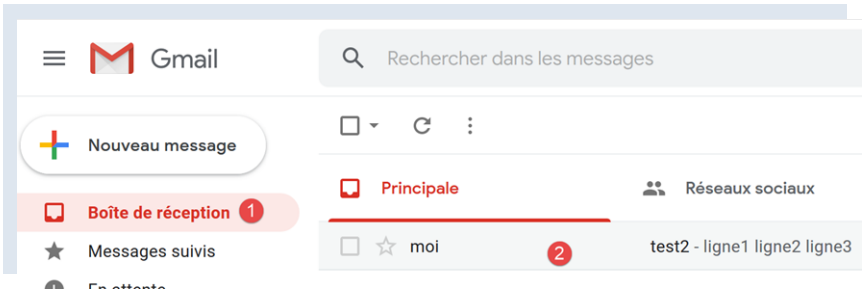
- en **[1]**, on se connecte sur le port 25 de la machine locale, là où opère le service SMTP de **[hMailServer]**. l'argument **[--quit bye]** indique que l'utilisateur quittera le programme en tapant la commande **[bye]**. Sans cet argument, la commande de fin du programme est **[quit]**. Or **[quit]** est également une commande du protocole SMTP. Il nous faut donc éviter cette ambiguïté ;

- en **[2]**, le client est bien connecté ;
- en **[3]**, le client attend des commandes tapées au clavier ;
- en **[4]**, le serveur lui envoie son message de bienvenue ;

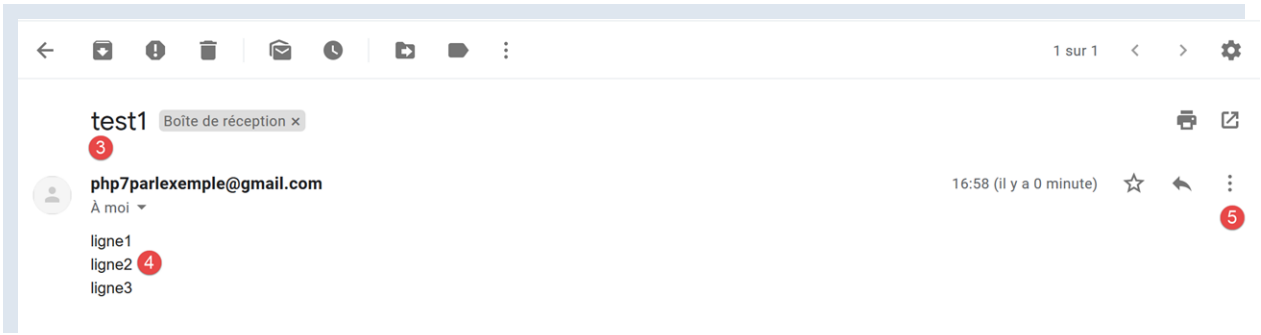
```
C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpClient.exe --quit bye localhost 25
Client [DESKTOP-528I5CU:50387] connecté au serveur [localhost-25]
Tapez vos commandes (bye pour arrêter) :
<-- [220 Bienvenue sur sergetahe@localhost]
EHLO localhost 5
<-- [250-DESKTOP-528I5CU]
<-- [250-SIZE 20480000]
<-- [250-AUTH LOGIN] 6
<-- [250 HELP]
MAIL FROM: <guest@localhost> 7
<-- [250 OK] 8
RCPT TO: <php7parlexemple@gmail.com> 9
<-- [250 OK] 10
DATA 11
<-- [354 OK, send.] 12
Subject: test2
From: <guest@localhost>  13
To: <php7parlexemple@gmail.com>
 14
ligne1
ligne2 15
ligne3
. 16
<-- [250 Queued (46.968 seconds)] 17
QUIT 18
<-- [221 goodbye] 19
Perte de la connexion avec le serveur... 20
```

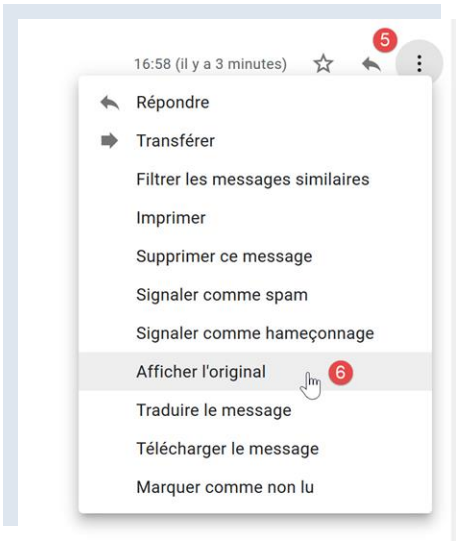
- en **[5]**, le client envoie la commande **[EHLO nom-de-la-machine-client]**. Le serveur lui répond par une suite de messages de la forme **[250-xx]** (6). Le code **[250]** indique le succès de la commande envoyée par le client ;
- en **[7]**, le client indique l'expéditeur du message, ici **[guest@localhost]**. Cet utilisateur doit exister sur le serveur de mails **[hMailServer]**. C'est le cas ici car nous avons créé cet utilisateur précédemment ;
- en **[8]**, la réponse du serveur ;
- en **[9]**, on indique le destinataire du message, ici l'utilisateur Gmail **[php7parlexemple@gmail.com]** ;
- en **[10]**, la réponse du serveur ;
- en **[11]**, la commande **[DATA]** indique au serveur que le client va envoyer le contenu du message ;
- en **[12]**, la réponse du serveur ;
- en **[13-16]**, le client doit envoyer une liste de lignes de texte terminée par une ligne ne contenant qu'un unique point. Le message peut contenir des lignes **[Subject :, From :, To :]** (13) pour définir respectivement le sujet du message, l'expéditeur, le destinataire ;
- en **[14]**, les entêtes précédents doivent être suivis d'une ligne vide ;
- en **[15]**, le texte du message ;
- en **[16]**, la ligne ne contenant qu'un unique point qui indique la fin du message ;
- en **[17]**, une fois que le serveur a reçu la ligne ne contenant qu'un unique point, il met le message en file d'attente ;
- en **[18]**, le client indique au serveur qu'il a fini ;
- en **[19]**, la réponse du serveur ;
- en **[20]**, on constate que le serveur a fermé la connexion qui le liait au client ;

Maintenant vérifions que l'utilisateur **[php7parlexemple@gmail.com]** a bien reçu le message :



- en **[2]**, on voit que l'utilisateur **[php7parlexemple@gmail.com]** a bien reçu le message ;





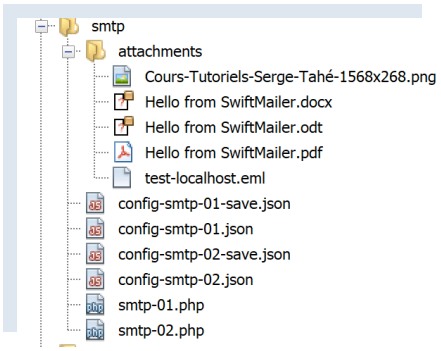

```
Return-Path: <php7parlexemple@gmail.com> 7
Received: from DESKTOP-528I5CU (lfbn-1-11924-110.w90-93.abo.wanadoo.fr. [90.93.230.110])
     8  by smtp.gmail.com with ESMTPSA id p8sm3172767wro.0.2019.05.19.07.58.15
        for <php7parlexemple@gmail.com>
        (version=TLS1_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);
        Sun, 19 May 2019 07:58:16 -0700 (PDT)
Date: Sun, 19 May 2019 07:58:16 -0700 (PDT)
From: php7parlexemple@gmail.com 9
X-Google-Original-From: <guest@localhost>10
Received: from localhost (localhost [127.0.0.1]) by DESKTOP-528I5CU with ESMTP ; Sun, 19 May 2019 16:58:13 +0200
Message-ID: <751775EA-5D46-45A6-BC9F-423EF4D36E28@DESKTOP-528I5CU>
Subject: test1 11
To: <php7parlexemple@gmail.com> 12

ligne1
ligne2 13
ligne3
```

- en **[7]**, l'expéditeur du mail. On voit que ce n'est pas **[guest@localhost]**. Cela est dû au fait que c'est le serveur relais défini dans la configuration de **[hmailServer]** qui a délivré le message. Or ce serveur relais est **[smtp.gmail.com]** associé aux identifiants de l'utilisateur Gmail **[php7parlexemple@gmail.com]**. Tout mail provenant de **[hMailServer]** semblera provenir de l'utilisateur **[php7parlexemple@gmail.com]**. Ce n'est pas ce qu'on voulait ici mais si on n'utilise pas ce serveur relais, le service SMTP de Gmail refuse les mails envoyés par **[hMailServer]** car le SMTP de Gmail réclame une authentification que **[hMailServer]** n'envoie pas. Il y a sans doute moyen de contourner ce problème mais je ne l'ai pas trouvé ;
- en **[8]**, on voit que le mail a été reçu de la machine **[DESKTOP-528I5CU]** qui héberge le serveur de mails **[hMailServer]** ;
- en **[9]**, l'expéditeur du message. On voit que ce n'est pas **[guest@localhost]** ;
- en **[10]**, l'expéditeur original du message. Cette fois-ci c'est bien **[guest@localhost]** ;
- en **[11]**, le sujet ;
- en **[12]**, le destinataire ;
- en **[13]**, le message ;

Finalement, notre client **[RawTcpClient]** a réussi à envoyer le message même si on a rencontré un problème pour l'expéditeur. Nous avons les bases pour créer un client SMTP écrit en PHP.

1.16.5.5 Un client SMTP basique écrit en PHP

Nous allons reproduire en PHP ce que nous avons appris précédemment du protocole SMTP.



Le script **[smtp-01.php]** est configuré par le fichier jSON **[config-smtp-01.json]** suivant :

```
{
  "mail to localhost via localhost": {
    "smtp-server": "localhost",
    "smtp-port": "25",
    "from": "guest@localhost",
    "to": "guest@localhost",
    "subject": "to localhost via localhost",
    "message": "ligne 1\nligne 2\nligne 3"
  },
  "mail to gmail via localhost": {
    "smtp-server": "localhost",
    "smtp-port": "25",
    "from": "guest@localhost",
    "to": "php7parlexemple@gmail.com",
    "subject": "to gmail via localhost",
    "message": "ligne 1\nligne 2\nligne 3"
  },
  "mail to gmail via gmail": {
    "smtp-server": "smtp.gmail.com",
    "smtp-port": "587",
    "from": "guest@localhost",
    "to": "php7parlexemple@gmail.com",
    "subject": "to gmail via gmail",
    "message": "ligne 1\nligne 2\nligne 3"
  }
}
```

[config-smtp-01.json] est un tableau où chacun des éléments est un dictionnaire de type **[nom=>infos]**. La valeur **[infos]** est elle-même un dictionnaire avec les clés et valeurs suivantes :

- **[smtp-server]** : le nom du serveur SMTP à utiliser ;
- **[smtp-port]** : le n° du port du service SMTP ;
- **[from]** : l'expéditeur du message ;
- **[to]** : le destinataire du message ;
- **[subject]** : le sujet du message ;
- **[message]** : le message à envoyer ;

- le 1er élément utilise le serveur SMTP **[localhost]** pour envoyer un mail à un utilisateur de **[localhost]** ;
- le 2d élément utilise le serveur SMTP **[localhost]** pour envoyer un mail à un utilisateur de **[Gmail]** ;
- le 3e élément utilise le serveur SMTP **[Gmail]** pour envoyer un mail à un utilisateur de **[Gmail]** ;

Le code **[smtp-01.php]** du client SMTP est le suivant :

```
<?php

// client SMTP (SendMail Transfer Protocol) permettant d'envoyer un message
// protocole de communication SMTP client-serveur
// -> client se connecte sur le port 25 du serveur smtp
// <- serveur lui envoie un message de bienvenue
// -> client envoie la commande EHLO nom de sa machine
```

```php
// <- serveur répond OK ou non
// -> client envoie la commande MAIL FROM: <expéditeur>
// <- serveur répond OK ou non
// -> client envoie la commande RCPT TO: <destinataire>
// <- serveur répond OK ou non
// -> client envoie la commande DATA
// <- serveur répond OK ou non
// -> client envoie ttes les lignes de son message et termine avec une ligne contenant le
// seul caractère .
// <- serveur répond OK ou non
// -> client envoie la commande QUIT
// <- serveur répond OK ou non
// les réponses du serveur ont la forme xxx texte où xxx est un nombre à 3 chiffres. Tout
// nombre xxx >=500 signale une erreur.
// La réponse peut comporter plusieurs lignes commençant toutes par xxx sauf la dernière
// de la forme xxx(espace)
// les lignes de texte échangées doivent se terminer par les caractères RC(#13) et LF(#10)
//
//  client SMTP (SendMail Transfer Protocol) permettant d'envoyer un message
//
// gestion des erreurs
//ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
//ini_set("display_errors", "off");
//
// respect strict des types déclarés des paramètres de foctions
declare (strict_types=1);
//
// les paramètres de l'envoi du courrier
const CONFIG_FILE_NAME = "config-smtp-01.json";

// on récupère la configuration
$mails = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true);

// envoi des courriers
foreach ($mails as $name => $infos) {
  // suivi
  print "Envoi du mail [$name]\n";
  // envoi du courrier
  $résultat = sendmail($name, $infos, TRUE);
  // affichage résultat
  print "$résultat\n";
}//for
// fin
exit;

//sendmail
//------------------------------------------------------------------------

function sendmail(string $name, array $infos, bool $verbose = TRUE): string {
  // envoie message[$name,$infos]. Si $verbose=TRUE      , fait un suivi des échanges client-serveur
  // on récupère le nom du client
  $client = gethostbyaddr(gethostbyname(""));
  // ouverture d'une connexion avec le serveur SMTP
  $connexion = fsockopen($infos["smtp-server"], (int) $infos["smtp-port"]);
  // retour si erreur
  if ($connexion === FALSE) {
    return sprintf("Echec de la connexion au site (%s,%s) : %s", $infos["smtp-server"], $infos["smtp-
port"]);
  }
  // $connexion représente un flux de communication bidirectionnel
  // entre le client (ce programme) et le serveur smtp contacté
  // ce canal est utilisé pour les échanges de commandes et d'informations
  // après la connexion le serveur envoie un message de bienvenue qu'on lit
  $erreur = sendCommand($connexion, "", $verbose, TRUE);
  if ($erreur !== "") {
    // fermeture de la connexion
    fclose($connexion);
    // retour
    return $erreur;
  }
  // cmde EHLO
  $erreur = sendCommand($connexion, "EHLO $client", $verbose, TRUE);
  if ($erreur !== "") {
    // fermeture de la connexion
    fclose($connexion);
    // retour
    return $erreur;
```

```
84    }
85    // cmde MAIL FROM:
86    $erreur = sendCommand($connexion, sprintf("MAIL FROM: <%s>", $infos["from"]), $verbose, TRUE);
87    if ($erreur !== "") {
88      // fermeture de la connexion
89      fclose($connexion);
90      // retour
91      return $erreur;
92    }
93    // cmde RCPT TO:
94    $erreur = sendCommand($connexion, sprintf("RCPT TO: <%s>", $infos["to"]), $verbose, TRUE);
95    if ($erreur !== "") {
96      // fermeture de la connexion
97      fclose($connexion);
98      // retour
99      return $erreur;
100   }
101   // cmde DATA
102   $erreur = sendCommand($connexion, "DATA", $verbose, TRUE);
103   if ($erreur !== "") {
104     // fermeture de la connexion
105     fclose($connexion);
106     // retour
107     return $erreur;
108   }
109   // préparation message à envoyer
110   // il doit contenir les lignes
111   // From: expéditeur
112   // To: destinataire
113   // Subject:
114   // ligne vide
115   // Message
116   // .
117   $data = sprintf("From: %s\r\nTo: %s\r\nSubject: %s\r\n\r\n%s\r\n.\r\n", $infos["from"], $infos["to"],
    $infos["subject"], $infos["message"]);
118   $erreur = sendCommand($connexion, $data, $verbose, FALSE);
119   if ($erreur !== "") {
120     // fermeture de la connexion
121     fclose($connexion);
122     // retour
123     return $erreur;
124   }
125   // cmde quit
126   $erreur = sendCommand($connexion, "QUIT", $verbose, TRUE);
127   if ($erreur !== "") {
128     // fermeture de la connexion
129     fclose($connexion);
130     // retour
131     return $erreur;
132   }
133   // fin
134   fclose($connexion);
135   return "Message envoyé";
136 }
137
138 // ------------------------------------------------------------------------
139
140 function sendCommand($connexion, string $commande, bool $verbose, bool $withRCLF): string {
141   // envoie $commande dans le canal $connexion
142   // mode verbeux si $verbose=1
143   // si $withRCLF=1, ajoute la séquence RCLF à échange
144   // données
145   if ($withRCLF) {
146     $RCLF = "\r\n";
147   } else {
148     $RCLF = "";
149   }
150   // envoi cmde si $commande non vide
151   if ($commande!=="") {
152     fputs($connexion, "$commande$RCLF");
153     // écho éventuel
154     if ($verbose) {
155       affiche($commande, 1);
156     }
157   }//if
158   // lecture réponse
159   $réponse = fgets($connexion, 1000);
```

```
160   // écho éventuel
161   if ($verbose) {
162     affiche($réponse, 2);
163   }
164   // récupération code erreur
165   $codeErreur = (int) substr($réponse, 0, 3);
166   // dernière ligne de la réponse ?
167   while (substr($réponse, 3, 1) === "-") {
168     // lecture réponse
169     $réponse = fgets($connexion, 1000);
170     // écho éventuel
171     if ($verbose) {
172       affiche($réponse, 2);
173     }
174   }//while
175   // réponse terminée
176   // erreur renvoyée par le serveur ?
177   if ($codeErreur >= 500) {
178     return substr($réponse, 4);
179   }
180 // retour sans erreur
181   return "";
182 }
183
184 // ------------------------------------------------------------------------
185
186 function affiche($échange, $sens) {
187   // affiche $échange à l'écran
188   // si $sens=1 affiche -->$echange
189   // si $sens=2 affiche <-- $échange sans les 2 derniers caractères RCLF
190   switch ($sens) {
191     case 1:
192       print "--> [$échange]\n";
193       break;
194     case 2:
195       $L = strlen($échange);
196       print "<-- [" . substr($échange, 0, $L - 2) . "]\n";
197       break;
198   }//switch
199 }
```

Commentaires

- ligne 39 : on exploite le fichier de configuration ;
- ligne 42 : on boucle sur les éléments du tableau **[mails]**. Chaque élément est un dictionnaire **[name=>infos]** où **[name]** est un nom qui peut être quelconque et **[infos]** un dictionnaire contenant les informations nécessaires à l'envoi d'un mail ;
- ligne 46 : l'envoi du mail est assuré par la fonction **[sendmail]** qui admet trois paramètres :
  - $name : le nom donné à cet envoi ;
  - $infos : le dictionnaire contenant les informations nécessaires à l'envoi ;
  - verbose : un booléen indiquant si les échanges client / serveur doivent être ou non logués sur la console ;
- ligne 46 : la fonction **[sendmail]** rend un message d'erreur qui est vide s'il n'y a pas eu d'erreur ;
- ligne 56 : la fonction **[sendmail]** envoie les différentes commandes que doit envoyer un client SMTP :
  - lignes 77-84 : la commande EHLO ;
  - lignes 85-92 : la commande MAIL FROM: ;
  - lignes 93-100 : la commande RCPT TO: ;
  - lignes 101-108 : la commande DATA ;
  - lignes 117-124 : envoi du message (From, To, Subject, texte) ;
  - lignes 125-132 : la commande QUIT ;
- ligne 140 : la fonction **[sendCommand]** est chargée d'envoyer les commandes du client au serveur SMTP. Elle admet quatre paramètres :
  - **[$connexion]** : la connexion qui relie le client au serveur ;
  - **[$commande]** : la commande à envoyer ;
  - **[$verbose]** : si TRUE alors les échanges client / serveur sont logués sur la console ;
  - **[$withRCLF]** : si TRUE, envoie la commande terminée par la séquence \r\n. C'est nécessaire pour toutes les commandes du protocole SMTP, mais **[sendCommand]** sert aussi à envoyer le message. Là on n'ajoute pas la séquence \r\n ;
- lignes 150-157 : la commande est envoyée au serveur ;
- lignes 158-163 : lecture de la 1re ligne de la réponse. Celle-ci peut comporter plusieurs lignes. Chaque ligne a la forme XXX-YYY où XXX est un code numérique sauf la dernière ligne de la réponse qui a la forme XXX YYY (absence du caractère -) ;
- lignes 167-174 : lecture de l'ensemble des lignes de la réponse ;
- ligne 177 : si le code numérique XXX est supérieur à 500, alors le serveur a renvoyé une erreur ;

Résultats

L'exécution du script donne les résultats console suivants :

```
1   Envoi du mail [mail to localhost via localhost]
2   <-- [220 Bienvenue sur sergetahe@localhost]
3   --> [EHLO DESKTOP-528I5CU.home]
4   <-- [250-DESKTOP-528I5CU]
5   <-- [250-SIZE 20480000]
6   <-- [250-AUTH LOGIN]
7   <-- [250 HELP]
8   --> [MAIL FROM: <guest@localhost>]
9   <-- [250 OK]
10  --> [RCPT TO: <guest@localhost>]
11  <-- [250 OK]
12  --> [DATA]
13  <-- [354 OK, send.]
14  --> [From: guest@localhost
15  To: guest@localhost
16  Subject: to localhost via localhost
17
18  ligne 1
19  ligne 2
20  ligne 3
21  .
22  ]
23  <-- [250 Queued (0.016 seconds)]
24  --> [QUIT]
25  <-- [221 goodbye]
26  Message envoyé
27  Envoi du mail [mail to gmail via localhost]
28  <-- [220 Bienvenue sur sergetahe@localhost]
29  --> [EHLO DESKTOP-528I5CU.home]
30  <-- [250-DESKTOP-528I5CU]
31  <-- [250-SIZE 20480000]
32  <-- [250-AUTH LOGIN]
33  <-- [250 HELP]
34  --> [MAIL FROM: <guest@localhost>]
35  <-- [250 OK]
36  --> [RCPT TO: <php7parlexemple@gmail.com>]
37  <-- [250 OK]
38  --> [DATA]
39  <-- [354 OK, send.]
40  --> [From: guest@localhost
41  To: php7parlexemple@gmail.com
42  Subject: to gmail via localhost
43
44  ligne 1
45  ligne 2
46  ligne 3
47  .
48  ]
49  <-- [250 Queued (0.000 seconds)]
50  --> [QUIT]
51  <-- [221 goodbye]
52  Message envoyé
53  Envoi du mail [mail to gmail via gmail]
54  <-- [220 smtp.gmail.com ESMTP d9sm21623375wro.26 - gsmtp]
55  --> [EHLO DESKTOP-528I5CU.home]
56  <-- [250-smtp.gmail.com at your service, [90.93.230.110]]
57  <-- [250-SIZE 35882577]
58  <-- [250-8BITMIME]
59  <-- [250-STARTTLS]
60  <-- [250-ENHANCEDSTATUSCODES]
61  <-- [250-PIPELINING]
62  <-- [250-CHUNKING]
63  <-- [250 SMTPUTF8]
64  --> [MAIL FROM: <guest@localhost>]
65  <-- [530 5.7.0 Must issue a STARTTLS command first. d9sm21623375wro.26 - gsmtp]
66  5.7.0 Must issue a STARTTLS command first. d9sm21623375wro.26 - gsmtp
67
68  Done.
```

- lignes 1-26 : l'utilisation du serveur SMTP **[hMailServer]** pour envoyer un mail à **[guest@localhost]** se passe bien ;
- lignes 27-52 : l'utilisation du serveur SMTP **[hMailServer]** pour envoyer un mail à **[php7parlexemple@gmail.com]** se passe bien ;

- lignes 53-65 : l'utilisation du serveur SMTP **[Gmail]** pour envoyer un mail à **[php7parlexemple@gmail.com]** ne se passe pas bien : en ligne 65, le serveur SMTP envoie un code d'erreur 530 avec le message d'erreur. Celui-ci indique que le client SMTP doit au préalable s'authentifier via une connexion sécurisée. Notre client ne l'a pas fait et est donc refusé ;

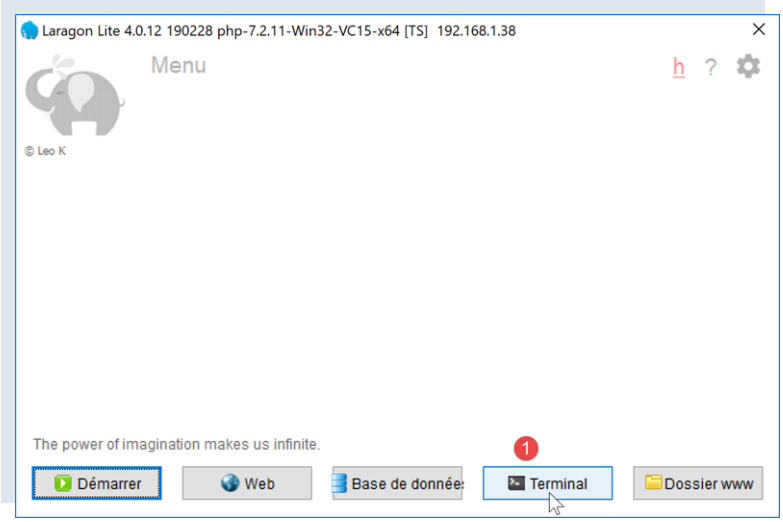
1.16.5.6 Un second client SMTP écrit avec la bibliothèque [SwiftMailer]

Le client précédent offre au moins deux insuffisances :

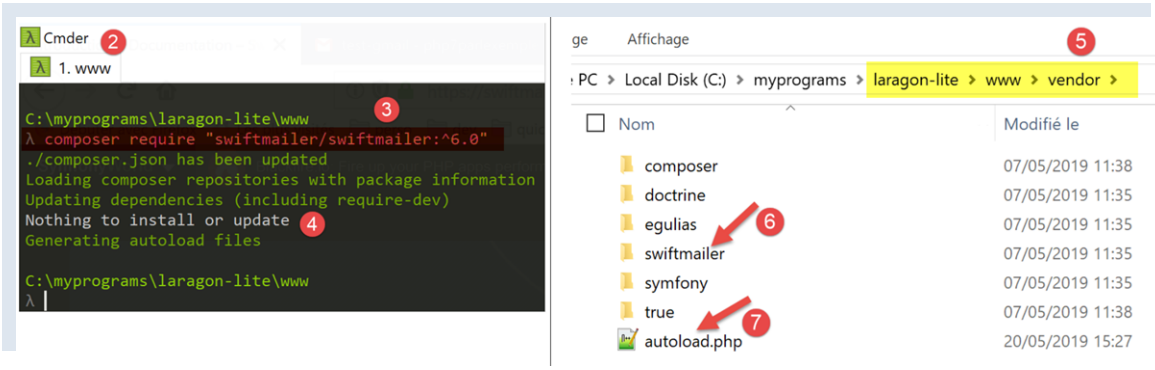
- il ne sait pas utiliser une connexion sécurisée si le serveur la réclame ;
- il ne sait pas joindre des attachements au message ;

Dans notre nouveau script nous allons utiliser la bibliothèque **[SwiftMailer]** **[https://swiftmailer.symfony.com/]** (mai 2019). Le mode d'installation de **[SwiftMailer]** est décrit à l'URL **[https://swiftmailer.symfony.com/docs/introduction.HTML]** (mai 2019).

Lancez tout d'abord Laragon :



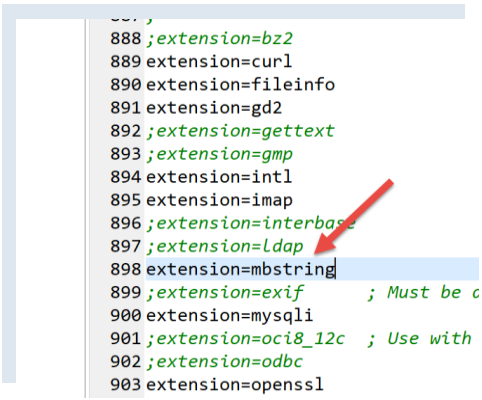
- en **[1]**, ouvrez un terminal ;



- en **[3]**, vérifiez que vous êtes dans le dossier **[<laragon>/www]** où <laragon> est le dossier d'installation de Laragon ;
- en **[3]**, tapez la commande indiquée (mai 2019). Vérifiez à l'URL **[https://swiftmailer.symfony.com/docs/introduction.html]** la commande exacte ;
- en **[4]**, il est indiqué qu'aucune installation ni mise à jour a été faite. C'est parce que la bibliothèque avait déjà été installée sur ce poste ;
- en **[5]**, le dossier d'installation de **[swiftmailer]** **[6]** ;
- en **[7]**, un fichier dont on aura besoin dans notre script ;

Ceci fait, vérifiez que le dossier **[<laragon>/www/vendor]** **[5]** est bien dans la branche **[Include Path]** de Netbeans (cf paragraphe lien).

Enfin la bibliothèque **[SwiftMailer]** nécessite que l'extension PHP **[mbstring]** soit active. Pour cela, on vérifie le fichier **[php.ini]** (cf paragraphe lien) :



Le script **[smtp-02.php]** utilisera le fichier de configuration jSON **[config-smtp-02.json]** suivant :

```
{
  "mail to localhost via localhost": {
    "smtp-server": "localhost",
    "smtp-port": "25",
    "from": "guest@localhost",
    "to": "guest@localhost",
    "subject": "test-localhost",
    "message": "ligne 1\nligne 2\nligne 3",
    "tls": "FALSE",
    "attachments": [
      "/attachments/Hello from SwiftMailer.docx",
      "/attachments/Hello from SwiftMailer.pdf",
      "/attachments/Hello from SwiftMailer.odt",
      "/attachments/Cours-Tutoriels-Serge-Tahé-1568x268.png",
      "/attachments/test-localhost.eml"
    ]
  },
  "mail to gmail via gmail": {
    "smtp-server": "smtp.gmail.com",
    "smtp-port": "587",
    "from": "php7parlexemple@gmail.com",
    "to": "php7parlexemple@gmail.com",
    "subject": "test-gmail-via-gmail",
    "message": "ligne 1\nligne 2\nligne 3",
    "tls": "TRUE",
    "user": "php7parlexemple@gmail.com",
    "password": "PHP7parlexemple",
    "attachments": [
      "/attachments/Hello from SwiftMailer.docx",
      "/attachments/Hello from SwiftMailer.pdf",
      "/attachments/Hello from SwiftMailer.odt",
      "/attachments/Cours-Tutoriels-Serge-Tahé-1568x268.png",
      "/attachments/test-localhost.eml"
    ]
  },
  "mail to gmail via localhost": {
    "smtp-server": "localhost",
    "smtp-port": "25",
    "from": "guest@localhost",
    "to": "php7parlexemple@gmail.com",
    "subject": "test-gmail-via-localhost",
    "message": "ligne 1\nligne 2\nligne 3",
    "tls": "FALSE",
    "attachments": [
      "/attachments/Hello from SwiftMailer.docx",
      "/attachments/Hello from SwiftMailer.pdf",
      "/attachments/Hello from SwiftMailer.odt",
      "/attachments/Cours-Tutoriels-Serge-Tahé-1568x268.png",
      "/attachments/test-localhost.eml"
    ]
  }
```

```
52. }
```

On retrouve les mêmes rubriques que dans le fichier **[config-smtp-01.json]** avec deux rubriques supplémentaires :

- **[tls]** : à TRUE indique qu'il faut utiliser une connexion sécurisée avec le serveur SMTP. Dans le cas où **[tls]** vaut TRUE, il faut ajouter deux rubriques :
  - **[user]** : le nom de l'utilisateur qui authentifie la connexion ;
  - **[password]** : son mot de passe ;
    - Dans notre exemple, nous avons utilisé les identifiants de l'utilisateur **[php7parlexemple@gmail.com]** pour nous connecter au serveur de Gmail. Utilisez les vôtres ;
- **[attachments]** : donne les noms des fichiers à attacher au mail ;

Le code du script **[smtp-02.php]** est le suivant :

```
1   <?php
2
3   // client SMTP (SendMail Transfer Protocol) permettant d'envoyer un message
4   //
5   // gestion des erreurs
6   //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7   //ini_set("display_errors", "off");
8   //
9   // dépendances
10  require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11  //
12  // les paramètres de l'envoi du courrier
13  const CONFIG_FILE_NAME = "config-smtp-02.json";
14
15  // on récupère la configuration
16  $mails = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true);
17
18  // envoi des courriers
19  foreach ($mails as $name => $infos) {
20    // suivi
21    print "Envoi du mail [$name]\n";
22    // envoi du courrier
23    $résultat = sendmail($name, $infos);
24    // affichage résultat
25    print "$résultat\n";
26  }//for
27  // fin
28  exit;
29
30  //---------------------------------------------------------------------
31
32  function sendmail($name, $infos) {
33
34    // envoie $infos[message] au serveur smtp $infos[smtp-server] sur le port $infos[smt-port]
35    // si $infos[tls] est vrai, le support TLS sera utilisé
36    // le mail est envoyé de la part de $infos[from]
37    // pour le destinataire $infos['to']
38    // Le document $info[attachment] est joint au message
39    // le message a le sujet $infos[subject]
40    //
41    // message au format HTML
42    $messageHTML = str_replace("\n", "<br/>", $infos["message"]);
43    try {
44      // création du message
45      $message = (new \Swift_Message())
46        // sujet du message
47        ->setSubject($infos["subject"])
48        // expéditeur
49        ->setFrom($infos["from"])
50        // destinataires avec un dictionnaire (setTo/setCc/setBcc)
51        ->setTo($infos["to"])
52        // texte du message
53        ->setBody($infos["message"])
54        // variante html
55        ->addPart("<b>$messageHTML</b>", 'text/html')
56      ;
57      // attachements
58      foreach ($infos["attachments"] as $attachment) {
59        // chemin de l'attachement
60        $fileName = __DIR__ . $attachment;
61        // on vérifie que le fichier existe
```

```
62        if (file_exists($fileName)) {
63          // on attache le document au message
64          $message->attach(\Swift_Attachment::fromPath($fileName));
65        } else {
66          // erreur
67          print "L'attachement [$fileName] n'existe pas\n";
68        }
69      }
70      // protocole TLS ?
71      if ($infos["tls"] === "TRUE") {
72        // TLS
73        $transport = (new \Swift_SmtpTransport($infos["smtp-server"], $infos["smtp-port"], 'tls'))
74          ->setUsername($infos["user"])
75          ->setPassword($infos["password"]);
76      } else {
77        // pas de TLS
78        $transport = (new \Swift_SmtpTransport($infos["smtp-server"], $infos["smtp-port"]));
79      }
80      // le gestionnaire de l'envoi
81      $mailer = new \Swift_Mailer($transport);
82      // envoi du message
83      $result = $mailer->send($message);
84      // fin
85      return "Message [$name] envoyé";
86    } catch (\Throwable $ex) {
87      // erreur
88      return "Erreur lors de l'envoi du message [$name] : " . $ex->getMessage();
89    }
90  }
```

Commentaires

- ligne 10 : nous chargeons le fichier **[autoload.php]** trouvé dans le dossier **[<lagagon>/www/vendor]** où <laragon> est le dossier d'installation de Laragon. Ce fichier va permettre de charger les fichiers de définition des classes de **[SwiftMailer]** dès le 1er usage de ces classes. Il nous évite de mettre autant de **[require]** que de classes et interfaces de SwiftMailer que nous allons utiliser ;
- ligne 32 : la nouvelle fonction **[sendmail]** qui a deux paramètres :
  - **[$name]** qui sert à différentier les messages entre-eux ;
  - **[$infos]** : les informations nécessaires pour envoyer le message à son destinataire ;
- ligne 42 : nous aurons deux versions du message : l'une en *plain text* l'autre en HTML. Ici, nous changeons les marques de fin de ligne en code HTML
 ;
- lignes 45-69 : nous définissons le message à l'aide de la classe **[\SwiftMessage]** ;
- ligne 47 : la méthode **[SwiftMessage→setSubject]** sert à fixer le sujet du message ;
- ligne 49 : la méthode **[SwiftMessage→setFrom]** sert à fixer l'expéditeur du message ;
- ligne 51 : la méthode **[SwiftMessage→setTo]** sert à fixer le destinataire du message ;
- ligne 53 : la méthode **[SwiftMessage→setBody]** sert à fixer le corps du message ;
- ligne 55 : la méthode **[SwiftMessage→addPart]** sert à fixer différentes versions du message, ici le message au format HTML. Lorsque le message a des variantes, les lecteurs de courrier affichent la variante préférée de l'utilisateur ;
- lignes 58-69 : la méthode **[SwiftMessage→addAttachment]** (64) permet d'attacher un fichier au message ;
- lignes 70-79 : une fois le message à envoyer défini, il faut définir comment l'envoyer. Le mode de transport du message est défini par la classe **[\Swift_SmtpTransport]**. Il y a au moins deux informations à fournir : le *nom* et le *port* du serveur SMTP. Il y en a également une troisième : le serveur SMTP impose-t-il une authentification sécurisée ?
- lignes 73-75 : l'instance **[\Swift_SmtpTransport]** pour une connexion sécurisée au serveur SMTP ;
- ligne 78 : l'instance **[\Swift_SmtpTransport]** pour une connexion non sécurisée au serveur SMTP ;
- ligne 81 : c'est la classe **[\SwiftMailer]** qui envoie les messages. On doit lui passer le mode de transport choisi ;
- ligne 83 : le message **[\SwiftMessage]** est envoyé par le biais du transport **[\Swift_SmtpTransport]** choisi. La méthode **[SwiftMailer→send]** rend le booléen FALSE si le message n'a pu être envoyé ;
- lignes 86-89 : la bibliothèque **[SwiftMailer]** lance une exception dès que quelque chose ne se passe pas bien ;

Note : on notera que l'espace de noms des classes de la bibliothèque **[SwiftMailer]** est la **racine **. On a explicitement noté les classes **[\SwiftMessage, \Swift_SmtpTransport, \SwiftMailer]** pour le rappeler ;

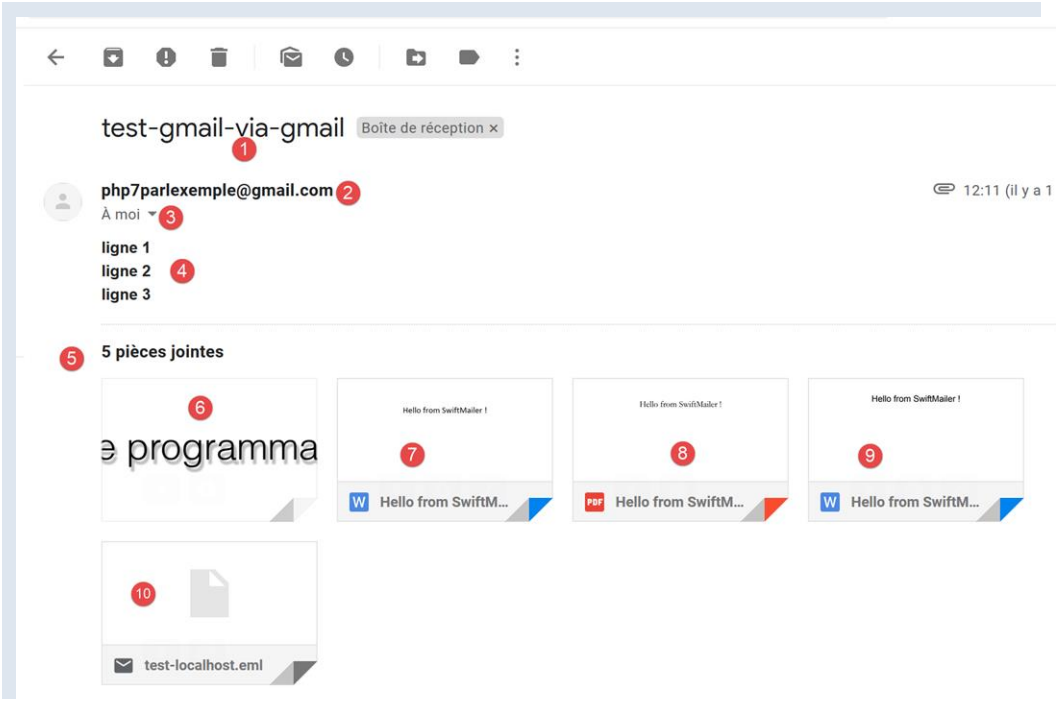
Résultats

Lorsqu'on exécute le script **[smtp-02.php]** on obtient les résultats console suivants :

```
1   Envoi du mail [mail to localhost via localhost]
2   Message [mail to localhost via localhost] envoyé
3   Envoi du mail [mail to gmail via gmail]
4   Message [mail to gmail via gmail] envoyé
```

```
5   Envoi du mail [mail to gmail via localhost]
6   Message [mail to gmail via localhost] envoyé
```

Si on consulte le compte Gmail de l'utilisateur **[php7parlexemple]** on a la chose suivante :



- en **[1]**, le sujet ;
- en **[2]**, l'expéditeur ;
- en **[3]**, le destinataire ;
- en **[4]**, le message ;
- en **[5-10]**, les pièces attachées ;

Si on demande à voir le message original, on obtient le document suivant :

```
1   Return-Path: <php7parlexemple@gmail.com>
2   Received: from [127.0.0.1] (lfbn-1-11924-110.w90-93.abo.wanadoo.fr. [90.93.230.110])
3           by smtp.gmail.com with ESMTPSA id e14sm7773816wma.41.2019.05.26.03.11.53
4           for <php7parlexemple@gmail.com>
5           (version=TLS1_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);
6           Sun, 26 May 2019 03:11:54 -0700 (PDT)
7   Message-ID: <e613c47a421a66e2cf7f8e319616ec49@swift.generated>
8   Date: Sun, 26 May 2019 10:11:53 +0000
9   Subject: test-gmail-via-gmail
10  From: php7parlexemple@gmail.com
11  To: php7parlexemple@gmail.com
12  MIME-Version: 1.0
13  Content-Type: multipart/mixed; boundary="_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_"
14
15  --_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_
16  Content-Type: multipart/alternative; boundary="_=_swift_1558865513_43c6d2a54065e4917fb06e3327f8d927_=_"
17
18  --_=_swift_1558865513_43c6d2a54065e4917fb06e3327f8d927_=_
19  Content-Type: text/plain; charset=utf-8
20  Content-Transfer-Encoding: quoted-printable
21
22  ligne 1
23  ligne 2
24  ligne 3
25
26  --_=_swift_1558865513_43c6d2a54065e4917fb06e3327f8d927_=_
27  Content-Type: text/HTML; charset=utf-8
28  Content-Transfer-Encoding: quoted-printable
29
30  <b>ligne 1<br/>ligne 2<br/>ligne 3</b>
31
```

```
--_=_swift_1558865513_43c6d2a54065e4917fb06e3327f8d927_=_--
--_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_
Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document; name="Hello from SwiftMailer.docx"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Hello from SwiftMailer.docx"


--_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_
Content-Type: application/pdf; name="Hello from SwiftMailer.pdf"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Hello from SwiftMailer.pdf"


--_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_
Content-Type: application/vnd.oasis.opendocument.text; name="Hello from SwiftMailer.odt"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Hello from SwiftMailer.odt"


--_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_
Content-Type: image/png; name="Cours-Tutoriels-Serge-Tahé-1568x268.png"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Cours-Tutoriels-Serge-Tahé-1568x268.png"


--_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_
Content-Type: message/rfc822; name=test-localhost.eml
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=test-localhost.eml

Return-Path: guest@localhost
Received: from [127.0.0.1] (localhost [127.0.0.1]) by DESKTOP-528I5CU with ESMTP ; Sat, 25 May 2019 09:48:23 +0200
Message-ID: <620f4628882b011feebe4faa30b45092@swift.generated>
Date: Sat, 25 May 2019 07:48:22 +0000
Subject: test-localhost
From: guest@localhost
To: guest@localhost
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="_=_swift_1558770502_c4b808c99c27ded04595bd11f4bad11b_=_"

--_=_swift_1558770502_c4b808c99c27ded04595bd11f4bad11b_=_
Content-Type: multipart/alternative; boundary="_=_swift_1558770503_3561ca315f33bd15ef6556e98db4a5b8_=_"

--_=_swift_1558770503_3561ca315f33bd15ef6556e98db4a5b8_=_
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: quoted-printable

j'ai =C3=A9t=C3=A9 invit=C3=A9 =C3=A0 d=C3=A9je=C3=BBner

--_=_swift_1558770503_3561ca315f33bd15ef6556e98db4a5b8_=_
Content-Type: text/HTML; charset=utf-8
Content-Transfer-Encoding: quoted-printable

<b>j'ai =C3=A9t=C3=A9 invit=C3=A9 =C3=A0 d=C3=A9je=C3=BBner</b>

--_=_swift_1558770503_3561ca315f33bd15ef6556e98db4a5b8_=_--
--_=_swift_1558770502_c4b808c99c27ded04595bd11f4bad11b_=_
Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document; name="Hello from SwiftMailer.docx"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Hello from SwiftMailer.docx"


--_=_swift_1558770502_c4b808c99c27ded04595bd11f4bad11b_=_
Content-Type: application/pdf; name="Hello from SwiftMailer.pdf"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Hello from SwiftMailer.pdf"


--_=_swift_1558770502_c4b808c99c27ded04595bd11f4bad11b_=_
Content-Type: application/vnd.oasis.opendocument.text; name="Hello from SwiftMailer.odt"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Hello from SwiftMailer.odt"


```

```
106 --_=_swift_1558770502_c4b808c99c27ded04595bd11f4bad11b_=_
107 Content-Type: image/png; name="Cours-Tutoriels-Serge-Tahé-1568x268.png"
108 Content-Transfer-Encoding: base64
109 Content-Disposition: attachment; filename="Cours-Tutoriels-Serge-Tahé-1568x268.png"
110
111
112 --_=_swift_1558770502_c4b808c99c27ded04595bd11f4bad11b_=_--
113
114 --_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_--
```

- ligne 9 : le sujet ;
- ligne 10 : l'expéditeur ;
- ligne 11 : le destinataire ;
- ligne 13 : le message contient plusieurs parties délimitées par des balises **[--_=_swift_xx]** ;
- lignes 19-24 : le message en *plain text* ;
- lignes 27-30 : le message en HTML ;
- lignes 34-36 : le fichier attaché **[Hello from SwiftMailer.docx]** ;
- lignes 40-42 : le fichier attaché **[Hello from SwiftMailer.pdf]** ;
- lignes 46-48 : le fichier attaché **[Hello from SwiftMailer.odt]** ;
- lignes 58-60 : le fichier attaché **[Cours-Tutoriels-Serge-Tahé-1568x268.png]** ;
- lignes 58-60 : le fichier attaché **[test-localhost.eml]** ;
- lignes 62-114 : le fichier attaché **[test-localhost.eml]** est lui-même un message dont le contenu est affiché aux lignes 62-114. On peut constater que ce message contient lui-même des attachements ;

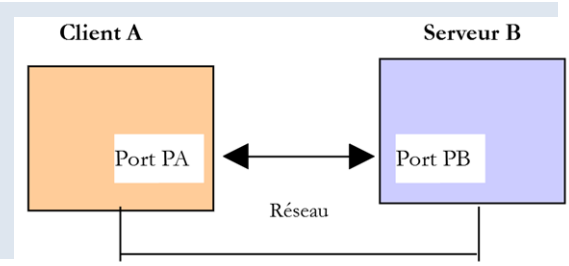
1.16.6 Les protocoles POP3 (Post Office Protocol) et IMAP (Internet Message Access Protocol)

1.16.6.1 Introduction

Pour lire les mails entreposés dans un serveur de mails, deux protocles existent :

- le protocole POP3 (Post Office Protocol) historiquement le 1er protocole mais peu utilisé maintenant ;
- le protocole IMAP (Internet Message Access Protocol) protocole plus récent que POP3 et le plus utilisé actuellement ;

Pour découvrir le protocole POP3, nous allons utiliser l'architecture suivante :



- **[Serveur B]** sera un serveur POP3 / IMAP local, implémenté par le serveur de mail **[hMailServer]** ;
- **[Client A]** sera un client POP3 / IMAP de diverses formes :
  - le client **[RawTcpClient]** pour découvrir le protocole POP3 ;
  - un script PHP rejouant le protocole POP3 du client **[RawTcpClient]** ;
  - un script PHP utilisant la bibliothèque IMAP de PHP qui permet d'implémenter aussi bien des clients IMAP que POP3 ;

1.16.6.2 Découverte du protocole POP3

Tout d'abord, nous utilisons le script **[smtp-01.php]** pour envoyer un mail à l'utilisateur **[guest@localhost]**. Si vous avez fait les tests associés au script, cet utilisateur a normalement reçu des mails mais on n'a pas pu le vérifier. Pour lui envoyer un nouveau mail, utilisez par exemple le fichier de configuration **[config-smtp-01.json]** suivant :

```
1.  {
2.    "mail to localhost via localhost": {
3.      "smtp-server": "localhost",
4.      "smtp-port": "25",
5.      "from": "guest@localhost",
6.      "to": "guest@localhost",
7.      "subject": "to localhost via localhost",
```

```
8.      "message": "ligne 1\nligne 2\nligne 3"
9.    }
10. }
```

Maintenant voyons avec le client **[RawTcpClient]** comment on peut lire la boîte mail de l'utilisateur **[guest@localhost]** :

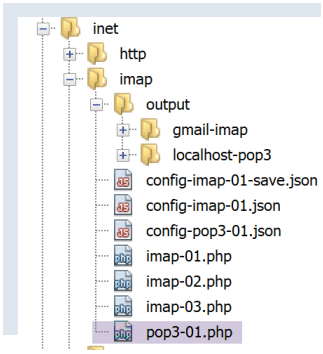
```
1   C:\Data\st-2019\dev\php7\php5-exemples\exemples\inet\utilitaires>RawTcpClient --quit bye localhost 110
2   Client [DESKTOP-528I5CU:55593] connecté au serveur [localhost-110]
3   Tapez vos commandes (bye pour arrêter) :
4   <-- [+OK Bienvenue sur sergetahe@localhost]
5   USER guest@localhost
6   <-- [+OK Send your password]
7   PASS guest
8   <-- [+OK Mailbox locked and ready]
9   LIST
10  <-- [+OK 2 messages (610 octets)]
11  <-- [1 305]
12  <-- [2 305]
13  <-- [.]
14  RETR 1
15  <-- [+OK 305 octets]
16  <-- [Return-Path: guest@localhost]
17  <-- [Received: from DESKTOP-528I5CU.home (localhost [127.0.0.1])]
18  <-- [   by DESKTOP-528I5CU with ESMTP]
19  <-- [   ; Tue, 21 May 2019 12:59:11 +0200]
20  <-- [Message-ID: <1356373A-33C9-4F31-BA43-2B119E128CE3@DESKTOP-528I5CU>]
21  <-- [From: guest@localhost]
22  <-- [To: guest@localhost]
23  <-- [Subject: to localhost via localhost]
24  <-- []
25  <-- [ligne 1]
26  <-- [ligne 2]
27  <-- [ligne 3]
28  <-- [.]
29  DELE 1
30  <-- [+OK msg deleted]
31  LIST
32  <-- [+OK 1 messages (305 octets)]
33  <-- [2 305]
34  <-- [.]
35  DELE 2
36  <-- [+OK msg deleted]
37  LIST
38  <-- [+OK 0 messages (0 octets)]
39  <-- [.]
40  QUIT
41  <-- [+OK POP3 server saying goodbye…]
42  Perte de la connexion avec le serveur…
```

- ligne 1 : le serveur POP3 travaille généralement avec le port 110. C'est le cas ici ;
- ligne 5 : la commande **[USER]** sert à définir l'utilisateur dont on veut lire la boîte mail ;
- ligne 7 : la commande **[PASS]** sert à définir son mot de passe ;
- ligne 9 : la commande **[LIST]** demande la liste des messages présents dans la boîte à lettres de l'utilisateur ;
- ligne 14 : la commande **[RETR]** demande à voir le message dont on passe le n° ;
- ligne 29 : la commande **[DELE]** demande la suppression du message dont on passe le n° ;
- ligne 40 : la commande **[QUIT]** indique au serveur qu'on a terminé ;

La réponse du serveur peut prendre plusieurs formes :

- une ligne unique commençant par **[+OK]** pour indiquer que la commande précédente du client a réussi ;
- une ligne unique commençant par **[-ERR]** pour indiquer que la commande précédente du client a échoué ;
- plusieurs lignes où :
  - la 1re ligne commence par **[+OK]** ;
  - la dernière ligne est constituée d'un unique point ;

1.16.6.3 Un script basique implémentant le protocole POP3



Comme le procole POP3 a la même structure que le protocole SMTP, le script **[pop3-01.php]** est un portage du script **[smtp-01.php]**. Il aura le fichier de configuration **[config-pop3-01.json]** suivant :

```
1.  {
2.    "localhost:110": {
3.      "server": "localhost",
4.      "port": "110",
5.      "user": "guest@localhost",
6.      "password": "guest",
7.      "maxmails": 5
8.    }
9.  }
```

- lignes 3-4 : le serveur POP3 interrogé est le serveur local **[hMailServer]** ;
- lignes 5-6 : on veut lire la boîte à lettres de l'utilisateur **[guest@localhost]** ;
- ligne 7 : on lira au plus 5 mails ;

Le script **[pop3-01.php]** est le suivant :

```
1   <?php
2
3   // client POP3 (Post Office Protocol) permettant de lire des messages d'une boîte à lettres
4   // protocole de co       mmunication POP3 client-serveur
5   // -> client se connecte sur le port 110 du serveur smtp
6   // <- serveur lui envoie un message de bienvenue
7   // -> client envoie la commande USER utilisateur
8   // <- serveur répond OK ou non
9   // -> client envoie la commande PASS mot_de_passe
10  // <- serveur répond OK ou non
11  // -> client envoie la commande LIST
12  // <- serveur répond OK ou non
13  // -> client envoie la commande RETR n° pour chacun des mails
14  // <- serveur répond OK ou non. Si OK envoie le contenu du mail demandé
15  // -> serveur envoie ttes les lignes du mail et termine avec une ligne contenant le
16  // seul caractère .
17  // -> client envoie la commande DELE n° pour supprimer un mail
18  // <- serveur répond OK ou non
19  // // -> client envoie la commande QUIT pour terminer le dialogue avec le serveur
20  // <- serveur répond OK ou non
21  // les réponses du serveur ont la forme +OK texte où -ERR texte
22  // La réponse peut comporter plusieurs lignes. Alors la dernière est constitué d'un unique point
23  // les lignes de texte échangées doivent se terminer par les caractères RC(#13) et LF(#10)
24  //
25  //  client POP3 (SendMail Transfer Protocol) permettant de lire des mails
26  //
27  // gestion des erreurs
28  //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
29  //ini_set("display_errors", "off");
30  //
31  // respect strict des types déclarés des paramètres de foctions
32  declare (strict_types=1);
33  //
34  // les paramètres de l'envoi du courrier
35  const CONFIG_FILE_NAME = "config-pop3-01.json";
36
37  // on récupère la configuration
```

```php
$mailboxes = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true);

// lecture des boîtes à lettres
foreach ($mailboxes as $name => $infos) {
  // suivi
  print "Lecture de la boîte à lettres [$name]\n";
  // lecture de la boîte à lettre
  $résultat = readmail($name, $infos, TRUE);
  // affichage résultat
  print "$résultat\n";
}//for
// fin
exit;

//readmail
//---------------------------------------------------------------------

function readmail(string $name, array $infos, bool $verbose = TRUE): string {
  // lit le contenu de la boîte à lettres [$name]
  // importe tous les messages
  // chaque message est supprimé aprèsb sa lecture
  // Si $verbose=1, fait un suivi des échanges client-serveur
  //
  // ouverture d'une connexion avec le serveur SMTP
  $connexion = fsockopen($infos["server"], (int) $infos["port"]);
  // retour si erreur
  if ($connexion === FALSE) {
    return sprintf("Echec de la connexion au site (%s,%s) : %s", $infos["smtp-server"], $infos["smtp-
port"]);
  }
  // $connexion représente un flux de communication bidirectionnel
  // entre le client (ce programme) et le serveur pop3 contacté
  // ce canal est utilisé pour les échanges de commandes et d'informations
  // après la connexion le serveur envoie un message de bienvenue qu'on lit
  $erreur = sendCommand($connexion, "", $verbose, TRUE);
  if ($erreur !== "") {
    // fermeture de la connexion
    fclose($connexion);
    // retour
    return $erreur;
  }
  // cmde USER
  $erreur = sendCommand($connexion, "USER {$infos["user"]}", $verbose, TRUE);
  if ($erreur !== "") {
    // fermeture de la connexion
    fclose($connexion);
    // retour
    return $erreur;
  }
  // cmde PASS
  $erreur = sendCommand($connexion, "PASS {$infos["password"]}", $verbose, TRUE);
  if ($erreur !== "") {
    // fermeture de la connexion
    fclose($connexion);
    // retour
    return $erreur;
  }
  // cmde LIST
  $premièreLigne = "";
  $erreur = sendCommand($connexion, "LIST", $verbose, TRUE, $premièreLigne);
  if ($erreur !== "") {
    // fermeture de la connexion
    fclose($connexion);
    // retour
    return $erreur;
  }
  // analyse de la 1re ligne pour connaître le nbre de messages
  $champs = [];
  preg_match("/^\+OK (\d+)/", $premièreLigne, $champs);
  $nbMessages = (int) $champs[1];
  // on boucle sur les messages
  $iMessage = 0;
  while ($iMessage < $nbMessages && $iMessage < $infos["maxmails"]) {
    // cmde RETR
    $erreur = sendCommand($connexion, "RETR " . ($iMessage + 1), $verbose, TRUE);
    if ($erreur !== "") {
      // fermeture de la connexion
```

```
114       fclose($connexion);
115       // retour
116       return $erreur;
117     }
118     // cmde DELE
119     $erreur = sendCommand($connexion, "DELE " . ($iMessage + 1), $verbose, TRUE);
120     if ($erreur !== "") {
121       // fermeture de la connexion
122       fclose($connexion);
123       // retour
124       return $erreur;
125     }
126     // msg suivant
127     $iMessage++;
128   }
129   // cmde QUIT
130   $erreur = sendCommand($connexion, "QUIT", $verbose, TRUE);
131   if ($erreur !== "") {
132     // fermeture de la connexion
133     fclose($connexion);
134     // retour
135     return $erreur;
136   }
137   // fin
138   fclose($connexion);
139   return "Terminé";
140 }
141
142 // ------------------------------------------------------------------------
143
144 function sendCommand($connexion, string $commande, bool $verbose, bool $withRCLF, string &$premièreLigne =
    ""): string {
145   // envoie $commande dans le canal $connexion
146   // mode verbeux si $verbose=1
147   // si $withRCLF=1, ajoute la séquence RCLF à échange
148   // met la 1re ligne de la réponse dans [$premièreLigne
149   // ]
150   // données
151   if ($withRCLF) {
152     $RCLF = "\r\n";
153   } else {
154     $RCLF = "";
155   }
156   // envoi cmde si $commande non vide
157   if ($commande !== "") {
158     fputs($connexion, "$commande$RCLF");
159     // écho éventuel
160     if ($verbose) {
161       affiche($commande, 1);
162     }
163   }//if
164   // lecture réponse
165   $réponse = fgets($connexion, 1000);
166   // on mémorise la 1re ligne
167   $premièreLigne = $réponse;
168   // écho éventuel
169   if ($verbose) {
170     affiche($réponse, 2);
171   }
172   // récupération code erreur
173   $codeErreur = substr($réponse, 0, 1);
174   if ($codeErreur === "-") {
175     // il y a eu une erreur
176     return substr($réponse, 5);
177   }
178   // cas particuliers des cmdes RETR et LIST qui ont des réponses à plusieurs lignes
179   $commande = substr(strtolower($commande), 0, 4);
180   if ($commande === "list" || $commande === "retr") {
181     // dernière ligne de la réponse ?
182     $champs = [];
183     $match = preg_match("/^\.\s+$/", $réponse, $champs);
184     while (!$match) {
185       // lecture réponse
186       $réponse = fgets($connexion, 1000);
187       // écho éventuel
188       if ($verbose) {
189         affiche($réponse, 2);
```

```
190       }
191       // analyse réponse
192       $champs = [];
193       $match = preg_match("/^\.\s+$/", $réponse, $champs);
194     }//while
195   }
196   // retour sans erreur
197   return "";
198 }
199
200 // ------------------------------------------------------------------------
201
202 function affiche($échange, $sens) {
203   // affiche $échange à l'écran
204   // si $sens=1 affiche -->$echange
205   // si $sens=2 affiche <-- $échange sans les 2 derniers caractères RCLF
206   switch ($sens) {
207     case 1:
208       print "--> [$échange]\n";
209       break;
210     case 2:
211       $L = strlen($échange);
212       print "<-- [" . substr($échange, 0, $L - 2) . "]\n";
213       break;
214   }//switch
215 }
```

Commentaires

Comme nous l'avons dit, **[pop3-01.php]** est un portage du script **[smtp-01.php]** que nous avons déjà commenté. Nous ne commenterons que les principales différences :

- ligne 55 : la fonction **[readmail]** est chargée de lire les mails de la boîte aux lettres. Les informations pour se connecter à cette boîte à lettres sont dans le dictionnaire **[$infos]** ;
- lignes 61-66 : ouverture d'une connexion avec le serveur POP3 ;
- lignes 71-77 : lecture du message de bienvenue envoyé par le serveur ;
- lignes 78-85 : on envoie la commande **[USER]** pour identifier l'utilisateur dont on veut les mails ;
- lignes 86-93 : on envoie la commande **[PASS]** pour donner le mot de passe de cet utilisateur ;
- lignes 94-102 : on envoie la commande **[LIST]** pour savoir combien il y a de mails dans la boîte à lettres de cet utilisateur.
- ligne 96 : on ajoute le paramètre **[$premièreLigne]** dans les paramètres de la fonction **[readmail]**. Dans la 1re ligne de sa réponse à la commande LIST, le serveur indique combien il y a de messages dans la boîte à lettres ;
- lignes 104-106 : on récupère le nombre de messages dans la 1re ligne de la réponse ;
- lignes 109-128 : on boucle sur chacun des messages. Pour chacun d'eux on émet deux commandes :
  - RETR i : pour récupérer le message n° i (lignes 111-117) ;
  - DELE i : pour le supprimer une fois qu'il a été lu (lignes 118-125) ;
- lignes 129-136 : on envoie la commande **[QUIT]** pour dire au serveur qu'on a terminé ;
- lignes 178-194 : pour les commandes **[LIST]** et **[RETR]**, la réponse du serveur a plusieurs lignes, la dernière étant constituée d'un unique point ;

Résultats

A l'exécution, on obtient les résultats suivants :

```
1   Lecture de la boîte à lettres [localhost:110]
2   <-- [+OK Bienvenue sur sergetahe@localhost]
3   --> [USER guest@localhost]
4   <-- [+OK Send your password]
5   --> [PASS guest]
6   <-- [+OK Mailbox locked and ready]
7   --> [LIST]
8   <-- [+OK 1 messages (305 octets)]
9   <-- [1 305]
10  <-- [.]
11  --> [RETR 1]
12  <-- [+OK 305 octets]
13  <-- [Return-Path: guest@localhost]
14  <-- [Received: from DESKTOP-528I5CU.home (localhost [127.0.0.1])]
15  <-- [    by DESKTOP-528I5CU with ESMTP]
16  <-- [    ; Tue, 21 May 2019 14:25:39 +0200]
17  <-- [Message-ID: <5F912826-F9C4-41B6-BDA7-4A29537781C9@DESKTOP-528I5CU>]
18  <-- [From: guest@localhost]
19  <-- [To: guest@localhost]
```

```
<-- [Subject: to localhost via localhost]
<-- []
<-- [ligne ]
<-- [ligne ]
<-- [ligne 3]
<-- [.]
--> [DELE 1]
<-- [+OK msg deleted]
--> [QUIT]
<-- [+OK POP3 server saying goodbye…]
Terminé
Done.
```

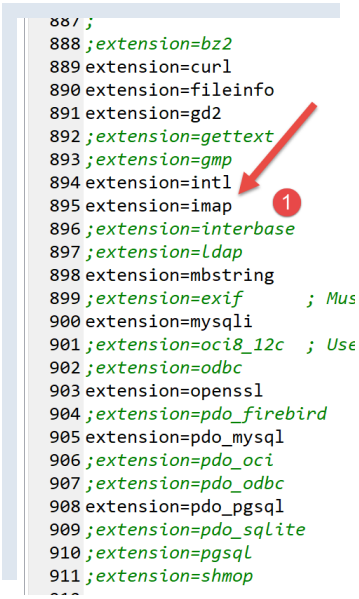
Nous avons là un client POP3 basique auquel il manque certaines capacités :

1 la possibilité de dialoguer avec un serveur POP3 sécurisé ;
2 la possibilité de lire les pièces attachées à un message ;

Nous allons implémenter la 1re possibilité avec les fonctions **[imap]** de PHP.

1.16.6.4 Client POP3 / IMAP implémenté avec les fonctions [imap] de PHP

Il nous faut tout d'abord vérifier que les fonctions **[imap]** sont disponibles dans la version de PHP que nous utilisons. Nous ouvrons le fichier **[php.ini]** décrit au paragraphe lien et nous cherchons les lignes qui parlent de **[imap]** :



Ligne 895, vérifiez que l'extension **[imap]** est bien activée.

Le script **[imap-01.php]** exploitera le fichier jSON **[config-imap-01.json]** suivant :

```
1.  {
2.    "{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX": {
3.      "imap-server": "imap.gmail.com",
4.      "imap-port": "993",
5.      "user": "php7parlexemple@gmail.com",
6.      "password": "PHP7parlexemple",
7.      "output-dir": "output/gmail-imap",
8.      "prefix": "message-"
9.    },
10.   "{localhost:110/pop3}": {
11.     "imap-server": "localhost",
12.     "imap-port": "110",
13.     "user": "guest@localhost",
14.     "password": "guest",
15.     "pop3": "TRUE",
16.     "output-dir": "output/localhost-pop3",
17.     "prefix": "message-"
18.   }
19. }
```

Le fichier **[config-imap-01.json]** définit un tableau de serveurs IMAP / POP3 à contacter. Chaque élément est une structure **[clé:valeur]**, où :

- **[clé]** : est le serveur à contacter. Nous en avons deux ici :
  - **[{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]** : désigne le serveur **[imap.gmail.com]** qui écoute sur le port 993. Le protocole client / serveur est IMAP. Le paramètre /ssl indique que la communication client / serveur est sécurisée. La paramètre /novalidate-cert demande au client de ne pas vérifier le certificat de sécurité que le serveur va lui envoyer. Enfin un serveur IMAP gère un ensemble de boîtes à lettres pour un même utilisateur. En précisant INBOX dans l'URL du serveur IMAP, nous indiquons que nous nous intéressons à la boîte à lettres nommée INBOX qui est normalement celle où arrivent les nouveaux messages ;
  - **[{localhost:110/pop3}INBOX]** : désigne le serveur **[localhost]** qui écoute sur le port 110. Le protocole client / serveur est ici POP3 ;
- **[valeur]** : est un dictionnaire précisant les points suivants :
  - **[imap-server]** : le nom du serveur IMAP ou POP3 ;
  - **[imap-port]** : le port du serveur IMAP ou POP3 ;
  - **[user]** : le propriétaire dont on veut lire la boîte à lettres ;
  - **[password]** : son mot de passe ;
  - **[output-dir]** : le dossier dans lequel on doit enregistrer les messages ;
  - **[prefix]** : le nom des fichiers où seront enregistrés les messages seront de la forme **prefixN** où N est un n° de message ;
  - **[pop3]** : un booléen à TRUE pour dire que le protocole utilisé est POP3. Dans ce cas après avoir lu un message, on le supprimera. C'est le fonctionnement habituel des serveurs POP3 : un message lu n'est pas conservé sur le serveur ;

Le script **[imap-01.php]** est le suivant :

```
1   <?php
2
3   // client IMAP (Internet Message Access Protocol) permettant de lire des mails
4   //
5   // respect strict des types déclarés des paramètres de foctions
6   declare (strict_types=1);
7   // gestion des erreurs
8   error_reporting(E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
9   //ini_set("display_errors", "off");
10  //
11  //
12  // les paramètres de lecture du courrier
13  const CONFIG_FILE_NAME = "config-imap-01.json";
14
15  // on récupère la configuration
16  $mailboxes = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true);
17
18  // lecture des boîtes à lettres
19  foreach ($mailboxes as $name => $infos) {
20    // suivi
21    print "------------Lecture de la boîte à lettres [$name]\n";
22    // lecture de la boîte à lettres
23    readmailbox($name, $infos);
24  }
25  // fin
26  exit;
27
28  //---------------------------------------------------------------------
29
30  function readmailbox(string $name, array $infos): void {
31    // Tentative de connexion
32    $imapResource = imap_open($name, $infos["user"], $infos["password"]);
33    // Test sur le retour de la fonction imap_open()
34    if (!$imapResource) {
35      // Échec
36      print "La connexion au serveur [$name] a échoué : " . imap_last_error() . "\n";
37    } else {
38      // Connexion établie
39      print "Connexion établie avec le serveur [$name].\n";
40      // total des messages dans la boîte à lettres
41      $nbmsg = imap_num_msg($imapResource);
42      print "Il y a [$nbmsg] messages dans la boîte à lettres [$name]\n";
43      // messages non lus dans la boîte aux lettres courante
44      if ($nbmsg > 0) {
45        print "Récupération de la liste des messages non lus de la boîte à lettres [$name]\n";
46        $msgNumbers = imap_search($imapResource, 'UNSEEN');
47        if ($msgNumbers === FALSE) {
48          print "Il n'y a pas de nouveaux messages dans la boîte à lettres [$name]\n";
49        } else {
50          foreach ($msgNumbers as $msgNumber) {
```

```
51            // on récupère des informations sur le message n° $msgNumber
52            $infosMail = imap_headerinfo($imapResource, $msgNumber);
53            if ($infosMail === FALSE) {
54              print "Statut du message n° [$msgNumber] de la boîte à lettres [$name] non récupéré : " .
    imap_last_error() . "\n";
55            } else {
56              print "Statut du message n° [$msgNumber] de la boîte à lettres [$name]\n";
57              print_r($infosMail);
58            }
59            // on récupère le corps du message n° $msgNumber
60            getMailBody($imapResource, $msgNumber, $infos);
61
62            // si le protocole est POP3, on supprime le message
63            $pop3 = $infos["pop3"];
64            if ($pop3 !== NULL) {
65              // on supprime le message en deux temps
66              imap_delete($imapResource, $msgNumber);
67              imap_expunge($imapResource);
68            }
69          }
70        }
71      }
72    }
73    // fermeture de la connexion
74    $imapClose = imap_close($imapResource);
75    if (!$imapClose) {
76      // Échec
77      print "La fermeture de la connexion a échoué : " . imap_last_error() . "\n";
78    } else {
79      // réussite
80      print "Fermeture de la connexion réussie.\n";
81    }
82  }
83
84  function getMailBody($imapResource, int $msgNumber, array $infos): void {
85    // on récupère le corps du message n° $msgNumber
86    $corpsMail = imap_body($imapResource, $msgNumber);
87
88    print "Enregistrement du message dans le fichier {$infos["output-dir"]}/{$infos["prefix"]}$msgNumber\n";
89    // on crée le dossier si besoin est
90    if (!file_exists($infos["output-dir"])) {
91      mkdir($infos["output-dir"]);
92    }
93    // on enregistre le message
94    if (!file_put_contents($infos["output-dir"] . "/" . $infos["prefix"] . $msgNumber, $corpsMail)) {
95      print "Echec de l'enregistrement\n";
96    }
97  }
```

Commentaires

- lignes 19-24 : on boucle sur l'ensemble des serveurs trouvés dans le fichier de configuration ;
- ligne 32 : la fonction **[raedmailbox]** lit la boîte à lettres indiqué dans **[$name]** ;
- ligne 32 : ouverture d'une connexion IMAP ;
  - le 1er paramètre est l'URL IMAP de la boîte à lettre à lire ;
  - le second paramètre est le nom de l'utilisateur propriétaire de cette boîte à lettres ;
  - le troisième paramètre est son mot de passe ;
    - La fonction **[imap_open]** organise la sécurisation de la connexion si l'URL IMAP de la boîte à lettres a le paramètre /ssl ;
- ligne 41 : la fonction **[imap_num_msg]** permet d'avoir le nombre total de messages de la boîte à lettres ;
- ligne 46 : la fonction **[imap_search]** permet de chercher certains messages. Ici, nous cherchons les messages qui n'ont pas encore été lus (UNSEEN). Le 2e paramètre est un critère de sélection. Il en existe une bonne vingtaine. La fonction **[imap_search]** rend un tableau de n°s de messages. Ceux-ci peuvent avoir deux formes : n° de séquence ou identifiant UID de message. Par défaut, La fonction **[imap_search]** rend un tableau de n°s de séquence. Si on ajoute un troisième paramètre **[SE_UID]** on aura les identifiants UID des messages ;
- ligne 47 : la fonction **[imap_search]** rend le booléen FALSE si elle n'a trouvé aucun message ;
- ligne 50 : on boucle sur tous les messages non lus ;
- ligne 52 : un message a des entêtes qu'on peut obtenir avec la fonction **[imap_headerinfo]**. Son 2e paramètre est normalement un n° de séquence de message. Si on veut mettre un identifiant UID de message, il faut mettre le 3e paramètre à **[FT_UID]** ;
- ligne 53 : la fonction **[imap_headerinfo]** rend le booléen FALSE si elle n'a pas pu faire son travail. Sinon elle rend un objet complexe qu'on affiche avec la fonction **[print_r]**, ligne 57 ;

- ligne 60 : après ses entêtes, on demande maintenant le corps du message avec la fonction **[imap_body]**. Cette fonction rend NULL si elle n'a pas pu faire son travail ;
- lignes 84-87 : on enregistre le corps du message dans un fichier local ;
- lignes 63-68 : si le protocole utilisé était POP3, on supprime le message qui vient d'être lu :
  - la fonction **[imap_delete]** marque le message comme « à supprimer » mais ne le supprime pas ;
  - la fonction **[imap_expunge]** supprime physiquement tous les messages qui ont été marqués « à supprimer » ;
- ligne 74 : on ferme la connexion avec le serveur IMAP. On utilise pour cela la fonction **[imap_close]** ;
- ligne 86 : la fonction **[imap_body]** permet d'avoir le corps d'un message repéré par son n° ;

Exécutons le script **[smtp-02.json]** pour que l'utilisateur **[php7parlexemple]** de Gmail et l'utilisateur **[guest]** de **[localhost]** aient de nouveaux messages. Ceci fait, exécutons le script **[imap-01.php]** pour lire leurs boîtes à lettres.

Les résultats console sont les suivants :

```
1   ------------Lecture de la boîte à lettres [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]
2   Connexion établie avec le serveur [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX].
3   Il y a [27] messages dans la boîte à lettres [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]
4   Récupération de la liste des messages non lus de la boîte à lettres
    [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]
5   Statut du message n° [26] de la boîte à lettres [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]
6   stdClass Object
7   (
8       [date] => Wed, 22 May 2019 10:08:24 +0000
9       [Date] => Wed, 22 May 2019 10:08:24 +0000
10      [subject] => test-gmail-via-gmail
11      [Subject] => test-gmail-via-gmail
12      [message_id] => <d8405cac62d57bd9c531ea79c146c72d@swift.generated>
13      [toaddress] => php7parlexemple@gmail.com
14      [to] => Array
15          (
16              [0] => stdClass Object
17                  (
18                      [mailbox] => php7parlexemple
19                      [host] => gmail.com
20                  )
21
22          )
23
24      [fromaddress] => php7parlexemple@gmail.com
25      [from] => Array
26          (
27              [0] => stdClass Object
28                  (
29                      [mailbox] => php7parlexemple
30                      [host] => gmail.com
31                  )
32
33          )
34
35      [reply_toaddress] => php7parlexemple@gmail.com
36      [reply_to] => Array
37          (
38              [0] => stdClass Object
39                  (
40                      [mailbox] => php7parlexemple
41                      [host] => gmail.com
42                  )
43
44          )
45
46      [senderaddress] => php7parlexemple@gmail.com
47      [sender] => Array
48          (
49              [0] => stdClass Object
50                  (
51                      [mailbox] => php7parlexemple
52                      [host] => gmail.com
53                  )
54
55          )
56
57      [Recent] =>
58      [Unseen] => U
59      [Flagged] =>
```

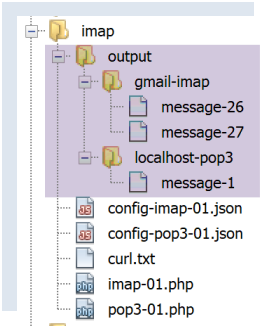
```
60      [Answered] =>
61      [Deleted] =>
62      [Draft] =>
63      [Msgno] =>   26
64      [MailDate] => 22-May-2019 10:08:29 +0000
65      [Size] => 19086
66      [udate] => 1558519709
67  )
68  Enregistrement du message dans le fichier output/gmail-imap/message-26
69  Statut du message n° [27] de la boîte à lettres [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]
70  stdClass Object
71  (
72      …
73  )
74  Enregistrement du message dans le fichier output/gmail-imap/message-27
75  Fermeture de la connexion réussie.
76  ------------Lecture de la boîte à lettres [{localhost:110/pop3}]
77  Connexion établie avec le serveur [{localhost:110/pop3}].
78  Il y a [1] messages dans la boîte à lettres [{localhost:110/pop3}]
79  Récupération de la liste des messages non lus de la boîte à lettres [{localhost:110/pop3}]
80  Statut du message n° [1] de la boîte à lettres [{localhost:110/pop3}]
81  stdClass Object
82  (
83      …
84  )
85  Enregistrement du message dans le fichier output/localhost-pop3/message-1
86  Fermeture de la connexion réussie.
87  Done.
```

Si aussitôt après ces résultats, nous réexécutons le script **[imap-01.php]**, les résultats sont alors les suivants :

```
1   ------------Lecture de la boîte à lettres [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]
2   Connexion établie avec le serveur [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX].
3   Il y a [27] messages dans la boîte à lettres [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]
4   Récupération de la liste des messages non lus de la boîte à lettres
    [{imap.gmail.com:993/imap/ssl/novalidate-cert}INBOX]
5   Il n'y a pas de nouveaux messages dans la boîte à lettres [{imap.gmail.com:993/imap/ssl/novalidate-
    cert}INBOX]
6   Fermeture de la connexion réussie.
7   ------------Lecture de la boîte à lettres [{localhost:110/pop3}]
8   Connexion établie avec le serveur [{localhost:110/pop3}].
9   Il y a [0] messages dans la boîte à lettres [{localhost:110/pop3}]
10  Fermeture de la connexion réussie.
```

- ligne 3 : il y a toujours le même nombre de messages dans la boîte à lettres Gmail mais il n'y a plus de nouveaux messages non lus (ligne 5). Ceci montre que l'exécution précédente a passé les messages lus du statut « non lu » au statut « lu » ;
- ligne 9 : il n'y a plus de messages dans la boîte à lettres de l'utilisateur **[guest@localhost]**. Cela vient du fait que dans l'exécution précédente, les messages lus sur **[localhost]** étaient ensuite supprimés ;

Les messages ont été enregistrés localement :



Si on regarde par-exemple le contenu du message n° 26 de Gmail, on a la chose suivante :

```
1
2   --_=_swift_1558519704_f31b373d6e416dc88eb4db0e45fb3a95_=_
3   Content-Type: multipart/alternative;
4    boundary="_=_swift_1558519706_9bffb48891232e50ab645383ca62242d_=_"
```

```
5
6
7   --_=_swift_1558519706_9bffb48891232e50ab645383ca62242d_=_
8   Content-Type: text/plain; charset=utf-8
9   Content-Transfer-Encoding: quoted-printable
10
11  ligne 1
12  ligne 2
13  ligne 3
14
15  --_=_swift_1558519706_9bffb48891232e50ab645383ca62242d_=_
16  Content-Type: text/HTML; charset=utf-8
17  Content-Transfer-Encoding: quoted-printable
18
19  <b>ligne 1<br/>ligne 2<br/>ligne 3</b>
20
21  --_=_swift_1558519706_9bffb48891232e50ab645383ca62242d_=_--
22
23
24  --_=_swift_1558519704_f31b373d6e416dc88eb4db0e45fb3a95_=_
25  Content-Type: application/pdf; name=Hello.pdf
26  Content-Transfer-Encoding: base64
27  Content-Disposition: attachment; filename=Hello.pdf
28
29  JVBERi0xLjUKJcOkw7zDtsOfCjIgMCBvYmoKPDwvTGVuZ3RoIDMgMCBSL0ZpbHRlci9GbGF0ZURl
30  Y29kZT4+CnN0cmVhbQp4nHWPuQoCQQyG+3mK1MKMyThHFoaAq7uF3cKAhdh5gIXgNr6+swcWshII
31  ..........
32  OTQwODU4RDUzRDVENjU0QzJCNTM3Mjc+IF0KL0RvY0NoZWNrc3VtIC9DMjU3MUY1MUNDRjgwQ0Ex
33  ODU0OUI0RTQ4NDkwMDM3OAo+PgpzdGFydHhyZWYKMTIzMjYKJSVFT0YK
34
35  --_=_swift_1558519704_f31b373d6e416dc88eb4db0e45fb3a95_=_--
36
```

- lignes 11-13 : le message en *plain text* ;
- ligne 19 : le message HTML ;
- ligne 25 : la pièce attachée ;

Essayons d'améliorer ce script pour avoir dans des fichiers séparés, les différents types de messages ainsi que les pièce attachées.

1.16.6.5 Client POP3 / IMAP amélioré

Dans le script **[imap-01.php]**, on affiche le corps du message n° i comme un fichier texte contenant à la fois les différents types de messages ainsi que le contenu codé des différentes pièce attachées. Il est possible d'obtenir la structure du message pour en connaître ces différentes parties. Dans le script **[imap-02.php]**, nous modifions la fonction **[getMailBody]** de la façon suivante :

```
1   function getMailBody($imapResource, int $msgNumber, array $infos): void {
2     // on récupère la structure du message
3     $structure=imap_fetchstructure($imapResource, $msgNumber);
4     // on l'affiche
5     print_r($structure);
6   }
```

- ligne 3 : nous demandons la structure du message ;
- ligne 5 : nous l'affichons ;

Le but est de connaître les informations contenues dans la structure d'un message pour voir comment on peut en obtenir les différentes parties. Dans notre exemple, le message est envoyé par le script **[smtp-02.php]** avec la configuration **[config-smtp-02.json]** suivante :

```
1.  {
2.    "mail to localhost via localhost": {
3.      "smtp-server": "localhost",
4.      "smtp-port": "25",
5.      "from": "guest@localhost",
6.      "to": "guest@localhost",
7.      "subject": "test-localhost",
8.      "message": "ligne 1\nligne 2\nligne 3",
9.      "tls": "FALSE",
10.     "attachments": [
11.       "/attachments/Hello from SwiftMailer.docx",
12.       "/attachments/Hello from SwiftMailer.pdf",
13.       "/attachments/Hello from SwiftMailer.odt",
14.       "/attachments/Cours-Tutoriels-Serge-Tahé-1568x268.png",
15.       "/attachments/test-localhost.eml"
```

```
16.     ]
17.   }
18. }
```

C'est donc un message avec cinq attachements qui est envoyé à **[guest@localhost]** (lignes 11-15). Le script **[imap-02.php]** est exécuté avec la configuration **[config-imap-01.json]** suivante :

```
1.  {
2.    "{localhost:110/pop3}": {
3.      "imap-server": "localhost",
4.      "imap-port": "110",
5.      "user": "guest@localhost",
6.      "password": "guest",
7.      "pop3": "TRUE",
8.      "output-dir": "output/localhost-pop3"
9.    }
10. }
```

C'est donc la boîte à lettres de **[guest@localhost]** qui est exploitée (ligne 5). Le script **[imap-02.php]** affiche alors la structure du message envoyé par **[smtp-02.php]**. Cette structure, affichée à la console, est la suivante :

```
1   stdClass Object
2   (
3       [type] => 1
4       [encoding] => 0
5       [ifsubtype] => 1
6       [subtype] => MIXED
7       [ifdescription] => 0
8       [ifid] => 0
9       [bytes] => 253599
10      [ifdisposition] => 0
11      [ifdparameters] => 0
12      [ifparameters] => 1
13      [parameters] => Array
14          (
15              [0] => stdClass Object
16                  (
17                      [attribute] => BOUNDARY
18                      [value] => _=_swift_1558872295_5bc8ee2ca8b3723c0b39ca8bbfbebdeb_=_
19                  )
20
21          )
22
23      [parts] => Array
24          (
25              [0] => stdClass Object
26                  (
27                      [type] => 1
28                      [encoding] => 0
29                      [ifsubtype] => 1
30                      [subtype] => ALTERNATIVE
31                      [ifdescription] => 0
32                      [ifid] => 0
33                      [bytes] => 429
34                      [ifdisposition] => 0
35                      [ifdparameters] => 0
36                      [ifparameters] => 1
37                      [parameters] => Array
38                          (
39                              [0] => stdClass Object
40                                  (
41                                      [attribute] => BOUNDARY
42                                      [value] => _=_swift_1558872296_1e51aae79dfca4e7e0af112489fe8734_=_
43                                  )
44
45                          )
46
47                      [parts] => Array
48                          (
49                              [0] => stdClass Object
50                                  (
51                                      [type] => 0
52                                      [encoding] => 4
53                                      [ifsubtype] => 1
54                                      [subtype] => PLAIN
55                                      [ifdescription] => 0
```

```
                                    [ifid] => 0
                                    [lines] => 3
                                    [bytes] => 27
                                    [ifdisposition] => 0
                                    [ifdparameters] => 0
                                    [ifparameters] => 1
                                    [parameters] => Array
                                        (
                                            [0] => stdClass Object
                                                (
                                                    [attribute] => CHARSET
                                                    [value] => utf-8
                                                )

                                        )

                                )

                            [1] => stdClass Object
                                (
                                    [type] => 0
                                    [encoding] => 4
                                    [ifsubtype] => 1
                                    [subtype] => HTML
                                    [ifdescription] => 0
                                    [ifid] => 0
                                    [lines] => 1
                                    [bytes] => 40
                                    [ifdisposition] => 0
                                    [ifdparameters] => 0
                                    [ifparameters] => 1
                                    [parameters] => Array
                                        (
                                            [0] => stdClass Object
                                                (
                                                    [attribute] => CHARSET
                                                    [value] => utf-8
                                                )

                                        )

                                )

                        )

                )

            [1] => stdClass Object
                (
                    [type] => 3
                    [encoding] => 3
                    [ifsubtype] => 1
                    [subtype] => VND.OPENXMLFORMATS-OFFICEDOCUMENT.WORDPROCESSINGML.DOCUMENT
                    [ifdescription] => 0
                    [ifid] => 0
                    [bytes] => 16302
                    [ifdisposition] => 1
                    [disposition] => ATTACHMENT
                    [ifdparameters] => 1
                    [dparameters] => Array
                        (
                            [0] => stdClass Object
                                (
                                    [attribute] => FILENAME
                                    [value] => Hello from SwiftMailer.docx
                                )

                        )

                    [ifparameters] => 1
                    [parameters] => Array
                        (
                            [0] => stdClass Object
                                (
                                    [attribute] => NAME
                                    [value] => Hello from SwiftMailer.docx
                                )
```

```
                        )

                )

            [2] => stdClass Object
                (
                    [type] => 3
                    [encoding] => 3
                    [ifsubtype] => 1
                    [subtype] => PDF
                    [ifdescription] => 0
                    [ifid] => 0
                    [bytes] => 17514
                    [ifdisposition] => 1
                    [disposition] => ATTACHMENT
                    [ifdparameters] => 1
                    [dparameters] => Array
                        (
                            [0] => stdClass Object
                                (
                                    [attribute] => FILENAME
                                    [value] => Hello from SwiftMailer.pdf
                                )

                        )

                    [ifparameters] => 1
                    [parameters] => Array
                        (
                            [0] => stdClass Object
                                (
                                    [attribute] => NAME
                                    [value] => Hello from SwiftMailer.pdf
                                )

                        )

                )

            [3] => stdClass Object
                (
                    …
                )

            [4] => stdClass Object
                (
                    …

                )

            [5] => stdClass Object
                (
                    [type] => 2
                    [encoding] => 3
                    [ifsubtype] => 1
                    [subtype] => RFC822
                    [ifdescription] => 0
                    [ifid] => 0
                    [lines] => 1881
                    [bytes] => 146682
                    [ifdisposition] => 1
                    [disposition] => ATTACHMENT
                    [ifdparameters] => 1
                    [dparameters] => Array
                        (
                            [0] => stdClass Object
                                (
                                    [attribute] => FILENAME
                                    [value] => test-localhost.eml
                                )

                        )

                    [ifparameters] => 1
                    [parameters] => Array
                        (
```

```
210                             [0] => stdClass Object
211                                 (
212                                     [attribute] => NAME
213                                     [value] => test-localhost.eml
214                                 )
215 
216                         )
217 
218                     [parts] => Array
219                         (
220                         …
221                         )
222 
223                 )
224 
225         )
226 
227 )
```

Commentaires

- la documentation PHP de la fonction **[imap_fetchstructure]** donne la signification des différents champs de l'objet retourné par la fonction :

Objets retournés par imap_fetchstructure()

type	Type primaire de corps
encoding	Codage de transfert du corps
ifsubtype	**TRUE** s'il y a une chaîne de sous type
subtype	sous type MIME
ifdescription	**TRUE** s'il y a une chaîne de description
description	Chaîne de description du contenu
ifid	**TRUE** s'il y a une chaîne d'identification
id	Chaîne d'identification
lines	Nombre de lignes
bytes	Nombre d'octets
ifdisposition	**TRUE** s'il y a une chaîne de disposition
disposition	Chaîne de disposition
ifdparameters	**TRUE** s'il y a un tableau de paramètres *dparameters*
dparameters	tableau d'objets où chaque objet a une propriété *"attribute"* et une propriété *"value"* correspondant aux paramètres d'en-têtes *Content-disposition* MIME.
ifparameters	**TRUE** si le tableau de paramètres existe
parameters	Tableau d'objets où chacun a une propriété *"attribute"* et une propriété *"value"*.
parts	Tableau d'objets décrivant chaque partie MIME du message

Les valeurs numériques du champ **[type]** ont la signification suivante :

Type primaire de corps (peut varier suivant la bibliothèque utilisée)

Valeur	Type	Constante
0	texte	TYPETEXT
1	multipart	TYPEMULTIPART
2	message	TYPEMESSAGE
3	application	TYPEAPPLICATION
4	audio	TYPEAUDIO
5	image	TYPEIMAGE
6	vidéo	TYPEVIDEO
7	modèle	TYPEMODEL
8	autre	TYPEOTHER

Les valeurs numériques du champ **[encoding]** ont la signification suivante :

Codage de transfert (peut varier suivant la bibliothèque utilisée)

Valeur	Type	Constante
0	7 bit	ENC7BIT
1	8 bit	ENC8BIT
2	Binaire	ENCBINARY
3	Base 64	ENCBASE64
4	Cité imprimable	ENCQUOTEDPRINTABLE
5	Autre	ENCOTHER

Le message enregistré par **[imap-01.php]** commençait par le texte suivant :

```
1   Return-Path: <php7parlexemple@gmail.com>
2   Received: from [127.0.0.1] (lfbn-1-11924-110.w90-93.abo.wanadoo.fr. [90.93.230.110])
3           by smtp.gmail.com with ESMTPSA id e14sm7773816wma.41.2019.05.26.03.11.53
4           for <php7parlexemple@gmail.com>
5           (version=TLS1_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);
6           Sun, 26 May 2019 03:11:54 -0700 (PDT)
7   Message-ID: <e613c47a421a66e2cf7f8e319616ec49@swift.generated>
8   Date: Sun, 26 May 2019 10:11:53 +0000
9   Subject: test-gmail-via-gmail
10  From: php7parlexemple@gmail.com
11  To: php7parlexemple@gmail.com
12  MIME-Version: 1.0
13  Content-Type: multipart/mixed; boundary="_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_"
14
15  --_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_
16  Content-Type: multipart/alternative; boundary="_=_swift_1558865513_43c6d2a54065e4917fb06e3327f8d927_=_"
17
18  --_=_swift_1558865513_43c6d2a54065e4917fb06e3327f8d927_=_
19  Content-Type: text/plain; charset=utf-8
20  Content-Transfer-Encoding: quoted-printable
21
22  ligne 1
23  ligne 2
24  ligne 3
25
26  --_=_swift_1558865513_43c6d2a54065e4917fb06e3327f8d927_=_
27  Content-Type: text/HTML; charset=utf-8
28  Content-Transfer-Encoding: quoted-printable
29
30  <b>ligne 1<br/>ligne 2<br/>ligne 3</b>
31
32  --_=_swift_1558865513_43c6d2a54065e4917fb06e3327f8d927_=_--
33  --_=_swift_1558865513_a3a939017128a4cfb867e968bce5df49_=_
```

- lignes 13 et 33 : délimitent le message de type **[multipart/mixed]** (ligne 13) ;
- lignes 16 et 32 : délimitent la partie de type **[multipart/alternative]** (ligne 16) ;
- lignes 18 et 26 : délimitent la 1re partie du message : le message en *plain text* ;
- lignes 26 et 32 : délimitent la seconde partie du message : le message HTML ;

Nous retrouvons les différentes informations du message ci-dessus dans l'objet retourné par **[imap_fetchstructure]** :

```
1   stdClass Object
2   (
3       [type] => 1
4       [encoding] => 0
5       [ifsubtype] => 1
6       [subtype] => MIXED
7       [ifdescription] => 0
8       [ifid] => 0
9       [bytes] => 253599
10      [ifdisposition] => 0
11      [ifdparameters] => 0
12      [ifparameters] => 1
13      [parameters] => Array
14          (
15              [0] => stdClass Object
16                  (
17                      [attribute] => BOUNDARY
18                      [value] => _=_swift_1558872295_5bc8ee2ca8b3723c0b39ca8bbfbebdeb_=_
19                  )
20
21          )
22
23      [parts] => Array
24          (
25              [0] => stdClass Object
26                  (
27                      [type] => 1
28                      [encoding] => 0
29                      [ifsubtype] => 1
30                      [subtype] => ALTERNATIVE
31                      [ifdescription] => 0
32                      [ifid] => 0
33                      [bytes] => 429
34                      [ifdisposition] => 0
35                      [ifdparameters] => 0
36                      [ifparameters] => 1
37                      [parameters] => Array
38                          (
39                              [0] => stdClass Object
40                                  (
41                                      [attribute] => BOUNDARY
42                                      [value] => _=_swift_1558872296_1e51aae79dfca4e7e0af112489fe8734_=_
43                                  )
44
45                          )
46
47                      [parts] => Array
48                          (
49                              [0] => stdClass Object
50                                  (
51                                      [type] => 0
52                                      [encoding] => 4
53                                      [ifsubtype] => 1
54                                      [subtype] => PLAIN
55                                      [ifdescription] => 0
56                                      [ifid] => 0
57                                      [lines] => 3
58                                      [bytes] => 27
59                                      [ifdisposition] => 0
60                                      [ifdparameters] => 0
61                                      [ifparameters] => 1
62                                      [parameters] => Array
63                                          (
64                                              [0] => stdClass Object
65                                                  (
66                                                      [attribute] => CHARSET
67                                                      [value] => utf-8
68                                                  )
```

```
69
70                                          )
71
72                                      )
73
74                                  [1] => stdClass Object
75                                      (
76                                          [type] => 0
77                                          [encoding] => 4
78                                          [ifsubtype] => 1
79                                          [subtype] => HTML
80                                          [ifdescription] => 0
81                                          [ifid] => 0
82                                          [lines] => 1
83                                          [bytes] => 40
84                                          [ifdisposition] => 0
85                                          [ifdparameters] => 0
86                                          [ifparameters] => 1
87                                          [parameters] => Array
88                                              (
89                                                  [0] => stdClass Object
90                                                      (
91                                                          [attribute] => CHARSET
92                                                          [value] => utf-8
93                                                      )
94
95                                              )
96
97                                      )
98
99                              )
100
101                 )
102
```

- ligne 3 : le message est de type MIME (Multipurpose Internet Mail Extensions) **[multipart]** ;
- ligne 4 : le message est encodé en 7 bits ;
- ligne 5 : **[ifsubtype]**=1 indique qu'il y a un champ **[subtype]** dans la structure ;
- ligne 6 : le champ **[subtype]** désigne un sous-type MIME, ici le type **[mixed]**. Au total le type MIME du document est **[multipart/mixed]** ;
- ligne 7 : **[ifdescription]**=0 indique qu'il n'y a pas de champ **[description]** dans la structure ;
- ligne 8 : **[ifid]**=0 indique qu'il n'y a pas de champ **[id]** dans la structure ;
- ligne 10 : **[ifdisposition]**=0 indique qu'il n'y a pas de champ **[disposition]** dans la structure ;
- ligne 11 : **[ifdparameters]**=0 indique qu'il n'y a pas de champ **[dparameters]** dans la structure ;
- ligne 12 : **[ifparameters]**=1 indique qu'il y a un champ **[parameters]** dans la structure ;
- ligne 13 : le champ **[parameters]** décrit les paramètres du message. Ici il n'y en a qu'un ;
- lignes 15-19 : cet objet décrit la ligne suivante du message texte :

```
boundary="_=_swift_1558872295_5bc8ee2ca8b3723c0b39ca8bbfbebdeb_=_"
```

  Ces lignes servent à délimiter le message. Dans le message récupéré par **[imap-01.php]**, la partie du message qui vient d'être décrite correspond à la ligne m). L'attribut **[boundary]** n'est pas le même car les copies d'écran correspondent au même message mais envoyé à des moments différents ;
- ligne 23 : commencent ici la structure des différentes parties du message ;
- lignes 25-45 : cette 1re partie est de type **[multipart/alternative]**. Elle correspond à la ligne p) du texte du message ;
- ligne 47 : cette 1re partie a elle-même des sous-parties ;
- lignes 47-70 : cette **1re sous-partie** est de type **[text/plain]** (lignes 51, 54), est encodée en type **[ENCQUOTEDPRINTABLE]** (ligne 52) et a un paramètre **[charset=utf-8]** (lignes 66-67) ;
- les lignes 49-72 décrivent les lignes s-x du message texte ;
- lignes 74-99 : décrivent la **seconde sous-partie** de la partie **[multipart/alternative]** ;
- lignes 74-99 : cette seconde sous-partie est de type **[text/HTML]** (lignes 76, 79), est encodée en type **[ENCQUOTEDPRINTABLE]** (ligne 77) et a un paramètre **[charset=utf-8]** (lignes 89-93) ;
- les lignes 74-99 décrivent les lignes aa-ad du message texte ;

La partie **[multipart/alternative]** est désormais terminée. Commence la partie **[application/vnd.openxmlformats-officedocument.wordprocessingml.document]** décrite par le texte suivant :

```
1   Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document; name="Hello from
    SwiftMailer.docx"
2   Content-Transfer-Encoding: base64
3   Content-Disposition: attachment; filename="Hello from SwiftMailer.docx"
```

Là encore, ces informations se retrouvent dans l'objet rapporté par la fonction **[imap_fetchstructure]** :

```
1   [1] => stdClass Object
2                   (
3                       [type] => 3
4                       [encoding] => 3
5                       [ifsubtype] => 1
6                       [subtype] => VND.OPENXMLFORMATS-OFFICEDOCUMENT.WORDPROCESSINGML.DOCUMENT
7                       [ifdescription] => 0
8                       [ifid] => 0
9                       [bytes] => 16302
10                      [ifdisposition] => 1
11                      [disposition] => ATTACHMENT
12                      [ifdparameters] => 1
13                      [dparameters] => Array
14                          (
15                              [0] => stdClass Object
16                                  (
17                                      [attribute] => FILENAME
18                                      [value] => Hello from SwiftMailer.docx
19                                  )
20
21                          )
22
23                      [ifparameters] => 1
24                      [parameters] => Array
25                          (
26                              [0] => stdClass Object
27                                  (
28                                      [attribute] => NAME
29                                      [value] => Hello from SwiftMailer.docx
30                                  )
31
32                          )
33
34                  )
35
36
```

- ligne 1 : c'est la seconde partie du message global. On rappelle que la 1re partie était de type **[multipart/alternative]** ;
- lignes 3-6 : cette seconde partie est de type **[application/vnd.openxmlformats-officedocument.wordprocessingml.document]** (lignes 3 et 6), est encodée en Base 64 (ligne 4) ;
- ligne 11 : cette seconde partie est une pièce attachée (ligne 11) et a deux paramètres : **[filename=Hello from SwiftMailer.docx]** (lignes 15-21) et **[name=Hello from SwiftMailer.docx]** (lignes 26-32). On remarquera que ce dernier paramètre n'existe pas dans le message texte. Il a donc été rajouté dans la fonction **[imap_fetchstructure]** ;

Les lignes 1-36 sont reproduites pour chacun des cinq attachements du message.

La fonction **[imap_fetch_structure]** nous permet donc d'avoir la structure d'un message. Celle-ci définit des parties qui elles-mêmes peuvent avoir des sous-parties. Pour avoir le texte d'une partie ou sous-partie on utilise la fonction **[imap_fetchbody]**.

Nous modifions la fonction **[getMailBody]** qui nous permet d'avoir le corps d'un message de la façon suivante :

```
1   function getMailBody($imapResource, int $msgNumber, array $infos, object $infosMail): void {
2     // on récupère la structure du message
3     $structure = imap_fetchstructure($imapResource, $msgNumber);
4     if ($structure !== FALSE) {
5       // on récupère ces différentes parties
6       getParts($imapResource, $msgNumber, $infos, $infosMail, $structure);
7     }
8   }
9
10  function getParts($imapResource, int $msgNumber, array $infos, object $infosMail, stdclass $part, string
    $sectionNumber = "0"): void {
11    // calcul du n° de section
12    if (substr($sectionNumber, 0, 2) === "0.") {
13      $sectionNumber = substr($sectionNumber, 2);
14    }
15    print "-----contenu de la partie n° [$sectionNumber]\n";
16    // type de contenu
17    print "Content-Type: ";
18    switch ($part->type) {
19      case TYPETEXT:
```

```
20        print "TEXT/{$part->subtype}\n";
21        break;
22      case TYPEMULTIPART:
23        print "MULTIPART/{$part->subtype}\n";
24        break;
25      case TYPEAPPLICATION:
26        print "APPLICATION/{$part->subtype}\n";
27        break;
28      case TYPEMESSAGE:
29        print "MESSAGE/{$part->subtype}\n";
30        break;
31      default:
32        print "UNKNOWN/{$part->subtype}\n";
33        break;
34    }
35    // type de codage
36    $encodings=["7 bits", "8 bits", "binaire", "base 64", "quoted-printable", "autre"];
37    print "Transfer-Encoding : ".$encodings[$part->encoding]."\n";
38
39    // on passe aux sous-parties éventuelles
40    if (isset($part->parts)) {
41      for ($i = 1; $i <= count($part->parts); $i++) {
42        // une nouvelle partie du message
43        $subpart = $part->parts[$i - 1];
44        // appel récursif - on demande le corps de la partie [$subpart]
45        getParts($imapResource, $msgNumber, $infos, $infosMail, $subpart, "$sectionNumber.$i");
46      }
47    }
48  }
```

Commentaires

- ligne 3 : nous récupérons la structure du message ;
- ligne 6 : nous demandons à voir ses différentes parties qui sont dans le tableau **[parts]** de la structure ;
- ligne 10 : la fonction **[getParts]** reçoit les paramètres suivants :
  - **[$imapResource]** : la connexion au serveur IMAP ;
  - **[$msgNumber]** : le n° de séquence du message dont on veut les parties ;
  - **[$infos]** : des informations pour savoir où stocker les parties qu'on va trouver, dans le système de fichiers local ;
  - **[$infosMail]** : des informations générales sur le mail (expéditeur, destinataire(s), sujet… ;
  - **[$part]** : un objet qui représente une partie du message ;
  - **[$sectionNumber]** : un n° de section (ou de partie) du message ;
- lignes 17-34 : on affiche le type de contenu de la partie n° **[$section]** du message. Pour cela on s'aide des champs **[$part→type]** et **[$part→subtype]** de la partie **[$part]** ;
- lignes 36-37 : on affiche le type de codage de la partie **[$sectionNumber]** ;
- lignes 40-47 : peut-être que la partie dont on vient d'afficher les informations a elle-même des sous-parties ;
- lignes 41-46 : si c'est le cas, on demande à voir le type de contenu des différentes sous-parties de la partie que l'on vient d'afficher. On fait ici, un appel récursif à la fonction **[getParts]** ;

De nouveau nous envoyons un mail à l'utilisateur Gmail **[php7parlexemple@gmail.com]** avec le script **[smtp-02.php]** et nous le lisons avec le script précédent **[imap-02.php]**. Cela donne les résultats console suivants :

```
1   ------------Lecture de la boîte à lettres [{localhost:110/pop3}]
2   Connexion établie avec le serveur [{localhost:110/pop3}].
3   Il y a [1] messages dans la boîte à lettres [{localhost:110/pop3}]
4   Récupération de la liste des messages non lus de la boîte à lettres [{localhost:110/pop3}]
5   -----contenu de la partie n° [0]
6   Content-Type: MULTIPART/MIXED
7   Transfer-Encoding : 7 bits
8   -----contenu de la partie n° [1]
9   Content-Type: MULTIPART/ALTERNATIVE
10  Transfer-Encoding : 7 bits
11  -----contenu de la partie n° [1.1]
12  Content-Type: TEXT/PLAIN
13  Transfer-Encoding : quoted-printable
14  -----contenu de la partie n° [1.2]
15  Content-Type: TEXT/HTML
16  Transfer-Encoding : quoted-printable
17  -----contenu de la partie n° [2]
18  Content-Type: APPLICATION/VND.OPENXMLFORMATS-OFFICEDOCUMENT.WORDPROCESSINGML.DOCUMENT
19  Transfer-Encoding : base 64
20  -----contenu de la partie n° [3]
21  Content-Type: APPLICATION/PDF
22  Transfer-Encoding : base 64
```

```
23  -----contenu de la partie n° [4]
24  Content-Type: APPLICATION/VND.OASIS.OPENDOCUMENT.TEXT
25  Transfer-Encoding : base 64
26  -----contenu de la partie n° [5]
27  Content-Type: UNKNOWN/PNG
28  Transfer-Encoding : base 64
29  -----contenu de la partie n° [6]
30  Content-Type: MESSAGE/RFC822
31  Transfer-Encoding : base 64
32  -----contenu de la partie n° [6.1]
33  Content-Type: TEXT/PLAIN
34  Transfer-Encoding : 7 bits
35  Fermeture de la connexion réussie.
```

On arrive bien à récupérer les différents types de contenu du message ainsi que leur type de codage. La numérotation des parties suit la règle suivante :

- lignes 6-7 : la partie **[multipart/mixed]** qui représente la totalité du message porte le n° 0. Les différentes parties de cet objet vont alors porter les n°s 1, 2…

Le message a au total cinq parties :

- lignes 9-10 : la partie **[multipart/alternative]** qui porte le n° 1 ;
- lignes 17-18 : la partie **[APPLICATION/VND.OPENXMLFORMATS-OFFICEDOCUMENT.WORDPROCESSINGML.DOCUMENT]** qui porte le n° 2. C'est l'attachement d'un fichier Word ;
- lignes 20-21 : la partie **[APPLICATION/PDF]** qui porte le n° 3. C'est l'attachement d'un fichier PDF ;
- lignes 23-24 : la partie **[APPLICATION/VND.OASIS.OPENDOCUMENT.TEXT]** qui porte le n° 4. C'est l'attachement d'un fichier OpenOffice ;
- lignes 26-27 : la partie **[UNKNOWN/PNG]** qui porte le n° 5. C'est l'attachement d'un fichier image ;
- lignes 30-31 : la partie **[MESSAGE/RFC822]** qui porte le n° 6. C'est l'attachement d'un mail ;

Lorsqu'une partie a des sous-parties, celles-ci sont numérotées x.1, x.2… où x est le n° de la partie englobante. Ainsi :

- lignes 11-12 : la 1re partie de la partie **[multipart/alternative]** porte le n° 1.1. C'est un contenu de type **[text/plain]** : le message du mail ;
- lignes 14-15 : la 2e partie de la partie **[multipart/alternative]** porte le n° 1.2. C'est un contenu de type **[text/HTML]** : le message du mail en HTML ;
- lignes 32-33 : la 1re partie de l'attachement **[MESSAGE/RFC822]** porte le n° 6.1. C'est un contenu de type **[text/plain]**. En fait selon le standard MIME, la numérotation des parties d'un attachement de mail **[MESSAGE/RFC822]** diffère de la règle décrite précédemment. Ainsi la 1re partie de l'attachement **[MESSAGE/RFC822]** ne porte pas le n° 6.1 mais un autre n° ;

Maintenant que nous savons comment repérer les différentes parties et sous-parties d'un mail, il nous reste à récupérer leur contenu.

Le code du script évolue de la façon suivante :

```
1   function getParts($imapResource, int $msgNumber, array $infos, object $infosMail, stdclass $part, string
    $sectionNumber = "0"): void {
2     // calcul du n° de section
3     if (substr($sectionNumber, 0, 2) === "0.") {
4       $sectionNumber = substr($sectionNumber, 2);
5     }
6     print "-----contenu de la partie n° [$sectionNumber]\n";
7     // type de contenu
8     print "Content-Type: ";
9     switch ($part->type) {
10      case TYPETEXT:
11        print "TEXT/{$part->subtype}\n";
12        break;
13      case TYPEMULTIPART:
14        print "MULTIPART/{$part->subtype}\n";
15        break;
16      case TYPEAPPLICATION:
17        print "APPLICATION/{$part->subtype}\n";
18        break;
19      case TYPEMESSAGE:
20        print "MESSAGE/{$part->subtype}\n";
21        break;
22      default:
23        print "UNKNOWN/{$part->subtype}\n";
24        break;
25    }
```

```
26    // type de codage
27    $encodings = ["7 bits", "8 bits", "binaire", "base 64", "quoted-printable", "autre"];
28    print "Transfer-Encoding : " . $encodings[$part->encoding] . "\n";
29
30    // est-ce un message ?
31    if ($part->type === TYPEMESSAGE) {
32      // on ne va pas gérer les sous-parties de ce message (mail attaché)
33      // on affiche le corps du mail attaché
34      print imap_fetchbody($imapResource, $msgNumber, $sectionNumber);
35    } else {
36      // on passe aux sous-parties éventuelles
37      if (isset($part->parts)) {
38        for ($i = 1; $i <= count($part->parts); $i++) {
39          // une nouvelle partie du message
40          $subpart = $part->parts[$i - 1];
41          // appel récursif - on demande le corps de la partie [$subpart]
42          getParts($imapResource, $msgNumber, $infos, $infosMail, $subpart, "$sectionNumber.$i");
43        }
44      } else {
45        // il n'y a pas de sous-parties - on affiche alors le corps du message
46        print imap_fetchbody($imapResource, $msgNumber, $sectionNumber);
47      }
48    }
49  }
```

Commentaires

- ligne 46 : la fonction **[imap_fetchbody]** récupère le corps de la partie n° **[$sectionNumber]** du message. La numérotation des parties d'un message suit la règle expliquée précédemment ;
- ligne 1 : on commence avec la section "0";
- ligne 41 : les sous-parties de cette section vont alors être numérotées "0.1", "0.2", alors qu'elles devraient être numérotées "1", "2"…
- lignes 3-5 : on corrige cette anomalie;
- lignes 37-43 : si la partie courante a des sous-parties, alors on boucle sur chacune d'elles (lignes 38-43). Leur n° de section est **[$sectionNumber.$i]** ;
- lignes 44-47 : lorsqu'il n'y a plus de sous-parties, on affiche le corps de la partie courante avec la fonction **[imap_fetchbody]**. Dans notre exemple, il s'agit des parties **[text/plain]**, **[text/HTML]** et les attachements ;

L'exécution de ce script donne les résultats suivants :

```
1   ------------Lecture de la boîte à lettres [{localhost:110/pop3}]
2   Connexion établie avec le serveur [{localhost:110/pop3}].
3   Il y a [1] messages dans la boîte à lettres [{localhost:110/pop3}]
4   Récupération de la liste des messages non lus de la boîte à lettres [{localhost:110/pop3}]
5   -----contenu de la partie n° [0]
6   Content-Type: MULTIPART/MIXED
7   Transfer-Encoding : 7 bits
8   -----contenu de la partie n° [1]
9   Content-Type: MULTIPART/ALTERNATIVE
10  Transfer-Encoding : 7 bits
11  -----contenu de la partie n° [1.1]
12  Content-Type: TEXT/PLAIN
13  Transfer-Encoding : quoted-printable
14  ligne 1
15  ligne 2
16  ligne 3
17  -----contenu de la partie n° [1.2]
18  Content-Type: TEXT/HTML
19  Transfer-Encoding : quoted-printable
20  <b>ligne 1<br/>ligne 2<br/>ligne 3</b>
21  -----contenu de la partie n° [2]
22  Content-Type: APPLICATION/VND.OPENXMLFORMATS-OFFICEDOCUMENT.WORDPROCESSINGML.DOCUMENT
23  Transfer-Encoding : base 64
24  UEsDBBQABgAIAAAAIQDfpNJsWgEAACAFAAATAAgCW0NvbnRlbnRfVHlwZXNdLnhtbCCiBAIooAAC
25  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
26  …
27  AAAAAAAAAF0mAABkb2NQcm9wcy9jb3JlLnhtbFBLAQItABQABgAIAAAAIQCdxkmwcgEAAMcCAAAQ
28  AAAAAAAAAAAAAAAAAAgpAABkb2NQcm9wcy9hcHAueG1sUEsFBgAAAAALAAsAwQIAALArAAAAAA==
29  -----contenu de la partie n° [3]
30  Content-Type: APPLICATION/PDF
31  Transfer-Encoding : base 64
32  JVBERi0xLjUKJcOkw7zDtsOfCjIgMCBvYmoKPDwvTGVuZ3RoIDMgMCBSL0ZpbHRlci9GbGF0ZURl
33  Y29kZT4+CnN0cmVhbQp4nHWNvQoCMRCE+zzF1sLF2WSTSyAEPD0Lu4OAhdj5AxaC1/j6Rk4s5GSa
34  …
```

```
35  PDcxQUJGQ0JGQURGODYxM0NBNUJDODNFMDNDNjI1QkQwPgo8NzFBQkZDQkZBREY4NjEzQ0E1QkM4
36  M0UwM0M2MjVCRDA+IF0KL0RvY0NoZWNrc3VtIC9DMTRCN0Q5N0YwNUU1OTYxQzhDODg0NEI3NkNF
37  OEIwRQo+PgpzdGFydHhyZWYKMTIzMTQKJSVFT0YK
38  -----contenu de la partie n° [4]
39  Content-Type: APPLICATION/VND.OASIS.OPENDOCUMENT.TEXT
40  Transfer-Encoding : base 64
41  UEsDBBQAAAgAAAs9uU5exjIMJwAAACcAAAAIAAAAbWltZXR5cGVhcHBsaWNhdGlvbi92bmQub2Fz
42  aXMub3BlbmRvY3VtZW50LnRleHRQSwMEFAAACAAACz25TgAAAAAAAAAAAAAAABwAAABDb25maWd1
43  …
44  AQIUABQACAgIAAs9uU42l0SORAQAABIRAAALAAAAAAAAAAAAAAAAI8bAABjb250ZW50LnhtbFBL
45  AQIUABQACAgIAAs9uU4Uf52+LgEAACUEAAAVAAAAAAAAAAAAAAAAAwgAABNRVRBLUlORi9tYW5p
46  ZmVzdC54bWxQSwUGAAAAABEAEQBlBAAAfSEAAAAA
47  -----contenu de la partie n° [5]
48  Content-Type: UNKNOWN/PNG
49  Transfer-Encoding : base 64
50  iVBORw0KGgoAAAANSUhEUgAABiAAAAEMCAYAAABN1n5OAAAACXBIWXMAAA7EAAAOxAGVKw4bAAAg
51  AElEQVR4nOy9e5TdV3Xn+Zm7aqprlBq1Rq1Wq7XU6opGrXaMMI6jAcfj9ihu4hAhkAghCCF0
52  …
53  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
54  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
55  AAAA2Mb8f9Q5r2ohJn6/AAAAAElFTkSuQmCC
56  -----contenu de la partie n° [6]
57  Content-Type: MESSAGE/RFC822
58  Transfer-Encoding : base 64
59  UmV0dXJuLVBhdGg6IGd1ZXN0QGxvY2FsaG9zdA0KUmVjZWl2ZWQ6IGZyb20gWzEyNy4wLjAuMV0g
60  KGxvY2FsaG9zdCBbMTI3LjAuMC4xXSkNCglieSBERVNLVE9QLTUyOEk1Q1Ugd2l0aCBFU01UUA0K
61  …
62  cjJvaEpuNi9BQUFBQUVsRlRrU3VRbUNDDQotLV89X3N3aWZ0XzE1NTg3NzA1MDJfYzRiODA4Yzk5
63  YzI3ZGVkMDQ1OTViZDExZjRiYWQxMWJfPV8tLQ0K
64  Fermeture de la connexion réussie.
```

Commentaires

- lignes 14-16 : le contenu du message texte codé en **[quoted-printable] (ligne 13)** ;
- ligne 20 : le contenu du message HTML codé en **[quoted-printable] (ligne 19)** ;
- lignes 24-28 : le contenu du fichier Word codé en **[base64] (ligne 23)** ;
- lignes 32-37 : le contenu du fichier PDF codé en **[base64] (ligne 31)** ;
- lignes 41-45 : le contenu du fichier OpenOffice codé en **[base64] (ligne 40)** ;
- lignes 50-55 ; le contenu du fichier image codé en **[base64] (ligne 49)** ;
- lignes 59-63 : le contenu du mail attaché codé en **[base64] ligne 58)** ;

Maintenant que :

- nous savons retrouver les textes des différentes parties d'un mail ;
- nous connaissons l'encodage de ces textes ;

nous pouvons sauvegarder ces textes dans des fichiers.

Le code évolue de la façon suivante :

```
1   function getParts($imapResource, int $msgNumber, array $infos, object $infosMail, stdclass $part, string
    $sectionNumber = "0"): void {
2     // calcul du n° de section
3     if (substr($sectionNumber, 0, 2) === "0.") {
4       $sectionNumber = substr($sectionNumber, 2);
5     }
6     print "-----contenu de la partie n° [$sectionNumber]\n";
7     // type de contenu
8     print "Content-Type: ";
9     switch ($part->type) {
10      case TYPETEXT:
11        print "TEXT/{$part->subtype}\n";
12        break;
13      case TYPEMULTIPART:
14        print "MULTIPART/{$part->subtype}\n";
15        break;
16      case TYPEAPPLICATION:
17        print "APPLICATION/{$part->subtype}\n";
18        break;
19      case TYPEMESSAGE:
20        print "MESSAGE/{$part->subtype}\n";
21        break;
22      default:
```

```
23          print "UNKNOWN/{$part->subtype}\n";
24          break;
25      }
26      // type de codage
27      $encodings = ["7 bits", "8 bits", "binaire", "base 64", "quoted-printable", "autre"];
28      print "Transfer-Encoding : " . $encodings[$part->encoding] . "\n";
29
30      // est-ce un message ?
31      if ($part->type === TYPEMESSAGE) {
32        // on ne va pas gérer les sous-parties de ce message
33        savePart($imapResource, $msgNumber, $sectionNumber, $infos, $infosMail);
34      } else {
35        // on passe aux sous-parties éventuelles
36        if (isset($part->parts)) {
37          for ($i = 1; $i <= count($part->parts); $i++) {
38            // une nouvelle partie du message
39            $subpart = $part->parts[$i - 1];
40            // appel récursif - on demande le corps de la partie [$subpart]
41            getParts($imapResource, $msgNumber, $infos, $infosMail, $subpart, "$sectionNumber.$i");
42          }
43        } else {
44          // il n'y a pas de sous-parties - on sauvegarde alors le corps du message
45          savePart($imapResource, $msgNumber, $sectionNumber, $infos, $infosMail);
46        }
47      }
48  }
```

- lignes 33 et 45 : l'affichage du texte d'une partie **[$imapResource, $msgNumber, $sectionNumber]** du mail est désormais remplacée par sa sauvegarde dans un fichier ;

La fonction **[savePart]** est la suivante :

```
1   // sauvegarde d'une partie de message
2   function savePart($imapResource, int $msgNumber, string $sectionNumber, array $infos, object $infosMail): void {
3     // dossier de sauvegarde
4     $outputDir = $infos["output-dir"] . "/message-$msgNumber";
5     // si le dossier n'existe pas, on le crée
6     if (!file_exists($outputDir)) {
7       mkdir($outputDir);
8     }
9     // structure de la partie à sauvegarder
10    $struct = imap_bodystruct($imapResource, $msgNumber, $sectionNumber);
11    // type de document
12    $type = $struct->type;
13    // sous-type de document
14    $subtype = "";
15    if (isset($struct->subtype)) {
16      $subtype = strtolower($struct->subtype);
17    }
18    // on analyse le type de la partie
19    switch ($type) {
20      case TYPETEXT:
21        // cas du message texte : text/xxx
22        switch ($subtype) {
23          case plain:
24            saveText("$outputDir/message.txt", 0, imap_fetchBody($imapResource, $msgNumber, $sectionNumber), $infosMail,
    $struct);
25            break;
26          case HTML:
27            saveText("$outputDir/message.HTML", 1, imap_fetchBody($imapResource, $msgNumber, $sectionNumber), $infosMail,
    $struct);
28            break;
29        }
30        break;
31      default:
32        // autres cas - on ne s'intéresse qu'aux attachements
33        if (isset($struct->disposition)) {
34          $disposition = strtolower($struct->disposition);
35          if ($disposition === "attachment") {
36            // on a affaire à un attachement - on le sauvegarde
37            saveAttachment($imapResource, $msgNumber, $sectionNumber, $outputDir, $struct);
38          }
39        } else {
40          // on ne traitera pas cette partie
41          print "Partie [$sectionNumber] ignorée\n";
42        }
43        break;
```

```
44    }
45  }
```

- lignes 3-8 : création du dossier de sauvegarde. Celui-ci porte le n° du message dont on analyse les parties ;
- ligne 10 : la partie de message à sauvegarder est définie de façon unique par les trois paramètres **[$imapResource, $msgNumber, $sectionNumber]**. On demande la structure de cette partie avec la fonction **[imap_bodystruct]** ;
- ligne 12 : on récupère le type principal de la partie de message ;
- lignes 13-17 : on récupère son sous-type ;
- lignes 20-30 : on traite les deux types de contenu : **[text/plain]** (lignes 23-25) et **[text/HTML]** (lignes 26-28). Les autres types **[text/xx]** sont ignorés ;
- ligne 24 : le texte de la partie **[text/plain]** sera sauvegardé dans un fichier **[message.txt]** ;
- ligne 27 : le texte de la partie **[text/HTML]** sera sauvegardé dans un fichier **[message.HTML]** ;
- lignes 31-43 : on traite le cas des parties dont le type principal n'est pas **[text]** ;
- ligne 35 : on ne s'intéresse qu'aux attachements du message ;
- ligne 37 : ceux-ci sont sauvegardés dans un fichier à l'aide de la fonction **[saveAttachment]** ;

Si on résume le code précédent :

- sauvegarde les parties **[text/plain]** et **[text/HTML]** à l'aide de la fonction **[saveText]**. Ces parties représentent le contenu du mail ;
- sauvegarde les différentes pièces attachées à l'aide de la fonction **[saveAttachment]** ;

La fonction **[saveText]** est la suivante :

```
1   // sauvegarde du texte [$text] du message
2   function saveText(string $fileName, int $type, string $text, object $infosMail, object $struct) {
3     // préparation du texte à sauvegarder
4     // $text est encodé - on le décode
5     switch ($struct->encoding) {
6       case ENCBASE64:
7         $text = base64_decode($text);
8         break;
9       case ENCQUOTEDPRINTABLE:
10        $text = quoted_printable_decode($text);
11        break;
12    }
13    // entêtes du message
14    // from
15    $from = "From: ";
16    foreach ($infosMail->from as $expéditeur) {
17      $from .= $expéditeur->mailbox . "@" . $expéditeur->host . ";";
18    }
19    // to
20    $to = "To: ";
21    foreach ($infosMail->to as $destinataire) {
22      $to .= $destinataire->mailbox . "@" . $destinataire->host . ";";
23    }
24    // subject
25    $subject = "Subject: " . $infosMail->subject;
26    // création du texte à enregistrer
27    switch ($type) {
28      case 0:
29        // text/plain
30        $contents = "$from\n$to\n$subject\n\n$text";
31        break;
32      case 1:
33        // text/HTML
34        $contents = "$from<br/>\n$to<br/>\n$subject<br/>\n<br/>\n$text";
35        break;
36    }
37    // création du fichier
38    print "sauvegarde d'un message dans [$fileName]\n";
39    // création du fichier
40    if (! file_put_contents($fileName, $contents)) {
41      // échec de la création du fichier
42      print "Impossible de créer le fichier [$fileName]\n";
43    }
44  }
```

Commentaires

- ligne 1 :

- **[$fileName]** est le nom du fichier dans lequel sera sauvegardé le texte **[$text]** ;
- **[$type]** : vaut 0 pour un fichier texte, 1 pour un fichier HTML ;
- **[$text]** : est le texte à sauvegarder. Mais il faut d'abord le décoder car il est codé ;
- **[$infosMail]** : contient des informations générales sur le mail. Nous allons utiliser les champs **[from, to, subject]** ;
- **[$struct]** : est la structure qui décrit la partie de mail qu'on est en train de sauvegarder. Cela va nous permettre de connaître le type d'encodage du texte à sauvegarder ;

- lignes 4-12 : on décode le texte à sauvegarder ;
- lignes 13-25 : on récupère les informations **[from, to, subject]** du mail ;
- lignes 27-36 : selon le type, 0 ou 1, du texte à sauvegarder, on construit un texte plain (ligne 30) ou un texte HTML (ligne 34) ;
- ligne 40 : le texte total est enregistré dans le fichier **[$fileName]** ;

Les attachements sont eux sauvegardés avec la fonction **[saveAttachment]** suivante :

```
1   // sauvegarde d'un attachement
2   function saveAttachment($imapResource, int $msgNumber, string $sectionNumber, string $outputDir, object
    $struct) {
3     // on analyse la structure de l'attachement
4     // on cherche à récupérer le nom du fichier dans lequel sauvegarder l'attachement
5     // ce nom se trouve dans les [dparameters] de la structure
6     if (isset($struct->dparameters)) {
7       // on récupère les [dparameters]
8       $dparameters = $struct->dparameters;
9       $fileName = "";
10      // on parcourt le tableau des [dparameters]
11      foreach ($dparameters as $dparameter) {
12        // chaque [dparameter] est un objet avec deux attributs [attribute, value]
13        $attribute = strtolower($dparameter->attribute);
14        // l'attribut [filename] correspond au nom du fichier à créer
15        // dans ce cas le nom du fichier est dans [$dparameter->value]
16        if ($attribute === "filename") {
17          $fileName = $dparameter->value;
18          break;
19        }
20      }
21      // si on n'a pas trouvé de nom de fichier, on regarde dans l'attribut [parameters] de la structure
22      if ($fileName === "" && isset($struct->parameters)) {
23        // on récupère les [parameters]
24        $parameters = $struct->parameters;
25        foreach ($parameters as $parameter) {
26          // chaque paramètre est un dictionnaire à deux clés [attribute, value]
27          $attribute = strtolower($parameter->attribute);
28          // si l'attribut est [name], alors le [value] est le nom du fichier
29          if ($attribute === "name") {
30            $fileName = $parameter->value;
31            // le nom du fichier peut être encodé
32            // par exemple =?utf-8?Q?Cours-Tutoriels-Serge-Tah=C3=A9-1568x268=2Ep
33            // on récupère l'encodage avec une expression régulière
34            $champs = [];
35            $match = preg_match("/=\?(.+?)\?/", $fileName, $champs);
36            // si concordance, alors on décode le nom du fichier
37            if ($match) {
38              $fileName = iconv_mime_decode($fileName, 0, $champs[1]);
39            }
40            break;
41          }
42        }
43      }
44    }
45    // si on a trouvé un nom de fichier, alors on sauvegarde l'attachement
46    if ($fileName !== "") {
47      // sauvegarde de l'attachement
48      $fileName = "$outputDir/$fileName";
49      print "sauvegarde de l'attachement dans [$fileName]\n";
50      // création fichier
51      if ($file = fopen($fileName, "w")) {
52        // on récupère le texte encodé de l'attachement
53        $text = imap_fetchbody($imapResource, $msgNumber, $sectionNumber);
54        // l'attachement est encodé - on le décode
55        switch ($struct->encoding) {
56          // base 64
57          case ENCBASE64:
58            $text = base64_decode($text);
59            break;
60          // quoted printable
61          case ENCQUOTEDPRINTABLE:
```

```
62            $text = quoted_printable_decode($text);
63            break;
64          default:
65            // on ignore les autres cas
66            break;
67        }
68        // écriture du texte dans le fichier
69        fputs($file, $text);
70        // fermeture fichier
71        fclose($file);
72      } else {
73        // échec de la création du fichier
74        print "L'attachement n'a pu être sauvegardé dans [$fileName]\n";
75      }
76    }
77  }
```

Commentaires

- ligne 2 : la fonction **[saveAttachment]** admet les paramètres suivants :
  - **[$imapResource, int $msgNumber, string $sectionNumber]** définissent de façon unique la partie IMAP à sauvegarder ;
  - **[string $outputDir]** est le dossier de sauvegarde ;
  - **[object $struct]** décrit la structure de la partie de message à sauvegarder ;
- lignes 6-44 : on cherche le nom du fichier associé à l'attachement. On utilisera ce même nom de fichier pour le sauvegarder. Le nom du fichier de l'attachement se trouve dans le tableau **[$struct→dparameters]** ou le tableau **[$struct→parameters]**, voire les deux ;
- lignes 30-40 : si le nom du fichier contient des caractères non codés sur 7 bits, alors il a été encodé en **[quoted-printable]**. Dans ce cas, dans **[$struct→dparameters]**, l'attribut s'appelle **[fileName*]** au lieu de **[fileName]**. Cela signifie qu'il n'a pas satisfait à la condition de la ligne 16. Le nom du fichier est alors cherché dans le tableau **[$struct→parameters]** ;
- ligne 32 : un exemple de nom de fichier encodé. Il a la forme suivante =?codage_original?codage_actuel?nom_encodé. Ainsi le nom **[=?utf-8?Q?Cours-Tutoriels-Serge-Tah=C3=A9-1568x268=2Ep]** signifie que le nom du fichier était en UTF-8 et qu'il est actuellement en **[quoted-printable]** (Q) ;
- ligne 38 : le nom du fichier est décodé avec la fonction **[iconv_mime_decode]** qui admet ici trois paramètres :
  - la chaîne à décoder ;
  - laisser à 0 par défaut ;
  - le jeu de caractères à utiliser pour représenter la chaîne décodée. Ce paramètre est présent dans la chaîne à décoder. On l'obtient avec une expression régulière aux lignes 34-35 ;
- lignes 45-75 : on sauvegarde l'attachement dans un fichier portant le nom qu'on a trouvé ;

Pour tester le script **[imap-02.php]**, on envoie d'abord un mail à **[guest@localhost]** avec la configuration suivante :

```
1.  {
2.    "mail to localhost via localhost": {
3.      "smtp-server": "localhost",
4.      "smtp-port": "25",
5.      "from": "guest@localhost",
6.      "to": "guest@localhost",
7.      "subject": "test-localhost",
8.      "message": "ligne 1\nligne 2\nligne 3",
9.      "tls": "FALSE",
10.     "attachments": [
11.       "/attachments/Hello from SwiftMailer.docx",
12.       "/attachments/Hello from SwiftMailer.pdf",
13.       "/attachments/Hello from SwiftMailer.odt",
14.       "/attachments/Cours-Tutoriels-Serge-Tahé-1568x268.png",
15.       "/attachments/test-localhost.eml"
16.     ]
17.   }
18. }
```

Il y a donc cinq attachements.

On lit le mail envoyé avec **[imap-02.php]** et la configuration suivante :

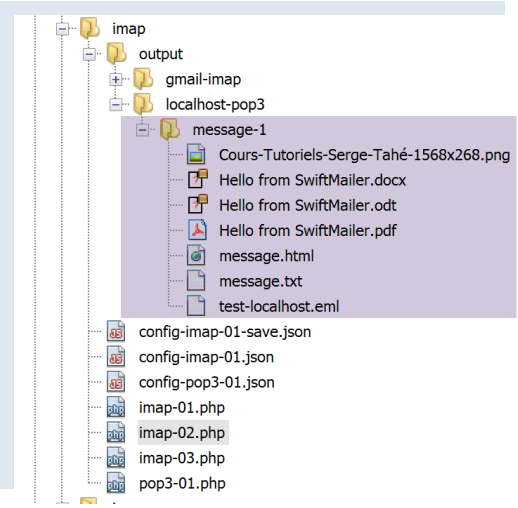
```
1.  {
2.    "{localhost:110/pop3}": {
3.      "imap-server": "localhost",
4.      "imap-port": "110",
5.      "user": "guest@localhost",
6.      "password": "guest",
```

```
7.      "pop3": "TRUE",
8.      "output-dir": "output/localhost-pop3"
9.    }
10. }
```

Les résultats console sont les suivants :

```
1   ------------Lecture de la boîte à lettres [{localhost:110/pop3}]
2   Connexion établie avec le serveur [{localhost:110/pop3}].
3   Il y a [1] messages dans la boîte à lettres [{localhost:110/pop3}]
4   Récupération de la liste des messages non lus de la boîte à lettres [{localhost:110/pop3}]
5   -----contenu de la partie n° [0]
6   Content-Type: MULTIPART/MIXED
7   Transfer-Encoding : 7 bits
8   -----contenu de la partie n° [1]
9   Content-Type: MULTIPART/ALTERNATIVE
10  Transfer-Encoding : 7 bits
11  -----contenu de la partie n° [1.1]
12  Content-Type: TEXT/PLAIN
13  Transfer-Encoding : quoted-printable
14  sauvegarde d'un message dans [output/localhost-pop3/message-1/message.txt]
15  -----contenu de la partie n° [1.2]
16  Content-Type: TEXT/HTML
17  Transfer-Encoding : quoted-printable
18  sauvegarde d'un message dans [output/localhost-pop3/message-1/message.HTML]
19  -----contenu de la partie n° [2]
20  Content-Type: APPLICATION/VND.OPENXMLFORMATS-OFFICEDOCUMENT.WORDPROCESSINGML.DOCUMENT
21  Transfer-Encoding : base 64
22  sauvegarde de l'attachement dans [output/localhost-pop3/message-1/Hello from SwiftMailer.docx]
23  -----contenu de la partie n° [3]
24  Content-Type: APPLICATION/PDF
25  Transfer-Encoding : base 64
26  sauvegarde de l'attachement dans [output/localhost-pop3/message-1/Hello from SwiftMailer.pdf]
27  -----contenu de la partie n° [4]
28  Content-Type: APPLICATION/VND.OASIS.OPENDOCUMENT.TEXT
29  Transfer-Encoding : base 64
30  sauvegarde de l'attachement dans [output/localhost-pop3/message-1/Hello from SwiftMailer.odt]
31  -----contenu de la partie n° [5]
32  Content-Type: UNKNOWN/PNG
33  Transfer-Encoding : base 64
34  sauvegarde de l'attachement dans [output/localhost-pop3/message-1/Cours-Tutoriels-Serge-Tahé-1568x268.png]
35  -----contenu de la partie n° [6]
36  Content-Type: MESSAGE/RFC822
37  Transfer-Encoding : base 64
38  sauvegarde de l'attachement dans [output/localhost-pop3/message-1/test-localhost.eml]
39  Fermeture de la connexion réussie.
40  Done.
```

On retrouve les fichiers sauvegardés dans le dossier **[output/localhost-pop3/message-N]** :



1.16.6.6 Client POP3 / IMAP avec la bibliothèque [php-mime-mail-parser]

Dans le script précédent **[imap-02.php]**, nous avons pu sauvegarder :

- les contenus **[text/plain]** et **[text/HTML]** du mail ;
- les attachements du mail ;

Pour un attachement de type **[message/rfc822]** nous avons également sauvegardé le contenu de l'attachement. Or ce type d'attachement est lui-même un mail qui, à son tour, a des contenus **[text/plain]** et **[text/HTML]** ainsi que des attachements. On peut alors être dans la situation suivante :

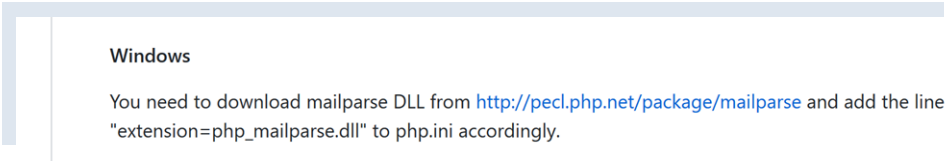
- un **[mail 1]** dont la structure est analogue celle d'un attachement de type **[message/rfc822]** ;
- un **[mail 2]** attaché au mail 1 ;
- un **[mail 3]** attaché au mail 2 ;
- etc…

Le script **[imap-02.php]** sauvegarde le contenu de **[mail 1]** (textes et attachements). Il sauvegarde **[mail 2]** comme document attaché mais s'arrête là. Il n'essaie pas d'analyser **[mail 2]** pour en extraire les textes et attachements. On pourrait penser qu'il suffit d'appliquer à **[mail 2]** ce qui a été fait pour **[mail 1]**. Un appel récursif à la méthode qui traitait **[mail 1]** pourrait alors suffire pour avoir le contenu de tous les mails imbriqués les uns dans les autres. Malheureusement les parties de **[mail 2]** sont numérotées avec une logique différente de celle utilisée pour **[mail 1]**, ce qui empêche d'utiliser le même algorithme dans les deux cas à moins d'utiliser une logique assez complexe pour calculer les n°s des parties d'un mail, quelque soit la position de celui-ci dans l'ensemble des mails imbriqués.

Le script **[imap-02.php]** était déjà complexe. Pour éviter de le complexifier encore davantage pour gérer les contenus des mails imbriqués, nous allons utiliser la bibliothèque **[php-mime-mail-parser]** disponible sur Github (mai 2019) à l'URL **[https://github.com/php-mime-mail-parser/php-mime-mail-parser]** et écrite par Vincent Dauce.

1.16.6.6.1 Installation de la bibliothèque [php-mime-mail-parser]

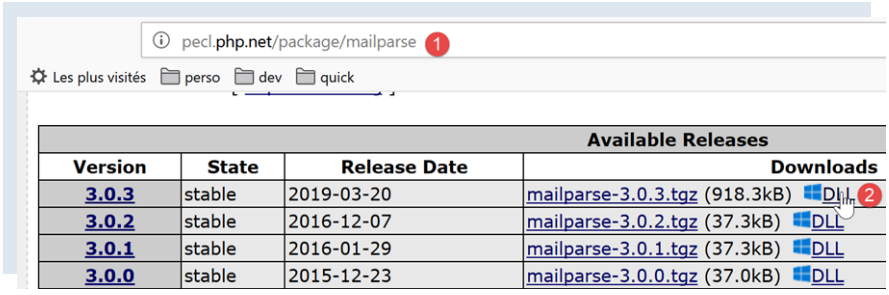
La page de présentation de la bibliothèque indique comment l'installer sous Windows :



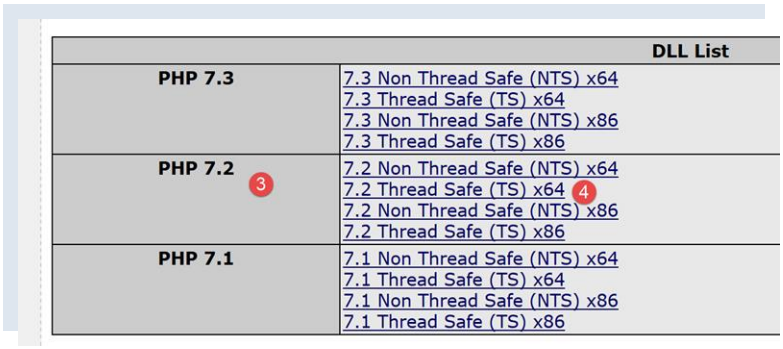
Il y a deux étapes pour l'OS Windows :

1 télécharger une DLL ;
2 modifier le fichier **[php.ini]** qui configure PHP ;

LA DLL de la bibliothèque **[mailparse]** est disponible à l'URL **[http://pecl.php.net/package/mailparse]** (mai 2019) ;



- en **[2]**, choisir la version la plus récente et stable de la bibliothèque ;



	DLL List
PHP 7.3	7.3 Non Thread Safe (NTS) x64 7.3 Thread Safe (TS) x64 7.3 Non Thread Safe (NTS) x86 7.3 Thread Safe (TS) x86
PHP 7.2 (3)	7.2 Non Thread Safe (NTS) x64 7.2 Thread Safe (TS) x64 (4) 7.2 Non Thread Safe (NTS) x86 7.2 Thread Safe (TS) x86
PHP 7.1	7.1 Non Thread Safe (NTS) x64 7.1 Thread Safe (TS) x64 7.1 Non Thread Safe (NTS) x86 7.1 Thread Safe (TS) x86

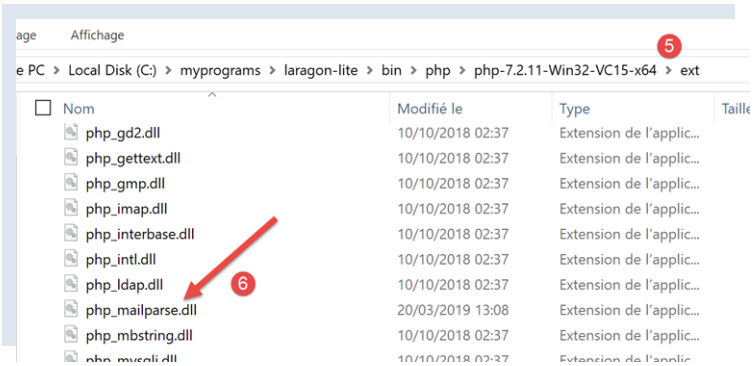
- en **[3]** choisir la version du PHP que vous utilisez (dans ce document c'est PHP 7.2) ;
- en **[4]**, choisir la version de votre OS Windows (ici c'est un Windows 64 bits). On prend la version **[Thread Safe]** ;

Pour connaître la version de PHP téléchargée avec Laragon, ouvrez un **[Terminal]** à partir de la fenêtre de Laragon et tapez la commande suivante :

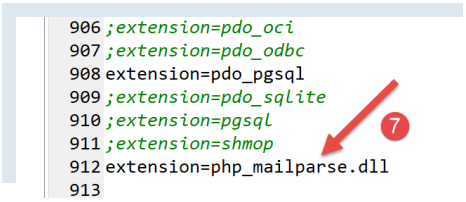
```
1   C:\myprograms\laragon-lite\www
2   λ php -v
3   PHP 7.2.11 (cli) (built: Oct 10 2018 02:04:07) ( ZTS MSVC15 (Visual C++ 2017) x64 )
4   Copyright (c) 1997-2018 The PHP Group
5   Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
```

La version de PHP 7.2.11 est donnée ligne 3. La même ligne donne la version de Windows utilisée pour la compilation (32 ou 74 bits).

Une fois la DLL obtenue, il faut la copier dans le dossier **[<laragon>/bin/php/<version-php>/ext] [5]** :



Ceci fait, il faut activer cette extension dans le fichier **[php.ini]** qui configure PHP (cf paragraphe lien) :



Il est probable que la ligne **[7]** n'existera pas et qu'il faudra l'ajouter par vous-même.

Une fois l'extension activée on peut vérifier sa validité en tapant la commande suivante dans un terminal de Laragon :

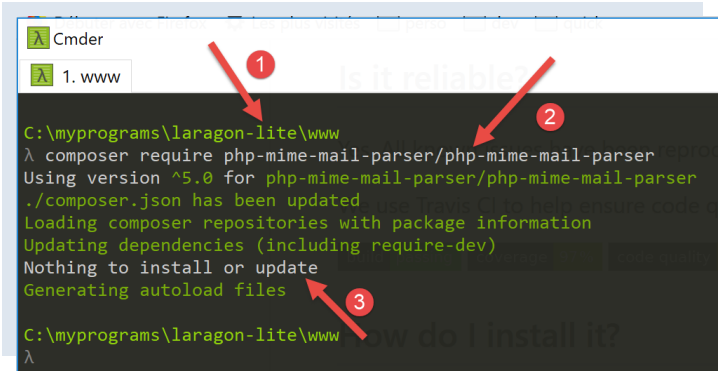
```
1   C:\myprograms\laragon-lite\www
2   λ php --ini
3   Configuration File (php.ini) Path: C:\windows
4   Loaded Configuration File:         C:\myprograms\laragon-lite\bin\php\php-7.2.11-Win32-VC15-x64\php.ini
5   Scan for additional .ini files in: (none)
6   Additional .ini files parsed:      (none)
```

La commande **[php --ini]** charge le fichier de configuration de la ligne 4. Elle va alors charger les DLL de toutes les extensions activées dans **[php.ini]**. Si l'une d'elles est erronée, cela sera signalé. Ainsi la validité de la DLL ajoutée **[php_mailparse.dll]** sera-t-elle vérifiée. Elle peut être déclarée incorrecte pour diverses raisons dont les plus fréquentes sont les suivantes :

- vous avez téléchargé une DLL ne correspondant pas à la version de PHP utilisée ;
- vous avez téléchargé une DLL 32 bits alors que vous avez un PHP 64 bits ou vice-versa ;

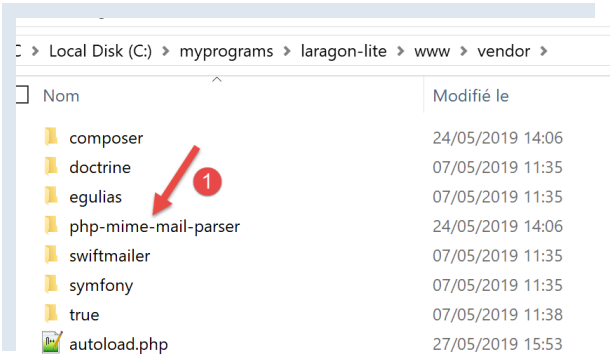
Une fois l'extension activée et vérifiée, on peut passer à l'installation de la bibliothèque **[php-mime-mail-parser]** :

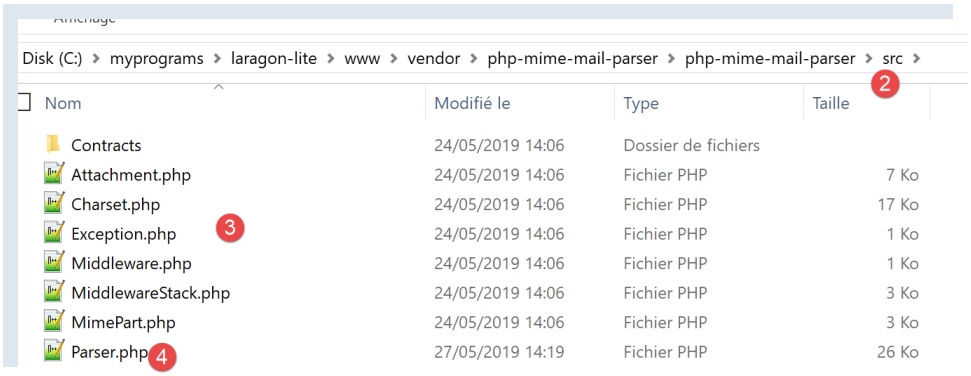
La commande **[8]** est à taper dans un terminal Laragon (cf paragraphe lien) :



- en **[1]**, vérifiez que vous êtes positionné sur le dossier **[<laragon>/www]** ;
- en **[2]**, la commande d'installation de la bibliothèque **[php-mime-mail-parser]** ;
- en **[3]**, rien n'a été installé ici car la bibliothèque **[php-mime-mail-parser]** était déjà installée ;

L'installation de bibliothèque **[php-mime-mail-parser]** se fait dans le dossier **[<laragon>/www/vendor]** :





- en **[2-3]**, les sources de la bibliothèque **[php-mime-mail-parser]** ;

Maintenant que l'environnement de travail a été installé, on peut passer à l'écriture du script **[imap-03.php]**.

1.16.6.6.2 Le script [imap-03.php]

Le script **[imap-03.php]** utilise le même fichier de configuration **[config-imap-01.json]** que les précédents scripts :

```
1.  {
2.    "{localhost:110/pop3}": {
3.      "imap-server": "localhost",
4.      "imap-port": "110",
5.      "user": "guest@localhost",
6.      "password": "guest",
7.      "pop3": "TRUE",
8.      "output-dir": "output/localhost-pop3"
9.    }
10. }
```

Le script **[imap-03.php]** est le suivant :

```
1   <?php
2
3   // client IMAP (Internet Message Access Protocol) permettant de lire des mails
4   // écrit avec la bibliothèque [php-mime-mail-parser]
5   // disponible à l'URL [https://github.com/php-mime-mail-parser/php-mime-mail-parser] (mai 2019)
6   //
7   // respect strict des types déclarés des paramètres de foctions
8   declare (strict_types=1);
9   // gestion des erreurs
10  error_reporting(E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
11  //ini_set("display_errors", "off");
12  //
13  // dépendances
14  require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
15  // les paramètres de lecture du courrier
16  const CONFIG_FILE_NAME = "config-imap-01.json";
17
18  // on récupère la configuration
19  if (!file_exists(CONFIG_FILE_NAME)) {
20    print "Le fichier de configuration " . CONFIG_FILE_NAME . " n'existe pas";
21    exit;
22  }
23  $mailboxes = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true);
24
25  // lecture des boîtes à lettres
26  foreach ($mailboxes as $name => $infos) {
27    // suivi
28    print "------------Lecture de la boîte à lettres [$name]\n";
29    // lecture de la boîte à lettres
30    readmailbox($name, $infos);
31  }
32  // fin
33  exit;
```

Commentaires

- lignes 18-23 : le contenu du fichier de configuration est mis dans le dictionnaire **[$mailboxes]** ;
- lignes 26-31 : chaque boîte à lettres est lue par la fonction **[readmailbox]** (ligne 30). Cette fonction lit en fait les messages non lus de la boîte à lettres. Une boîte à lettres correspond à l'adresse mail d'un utilisateur donné ;

La fonction **[readmailbox]** est la suivante :

```
1   function readmailbox(string $name, array $infos): void {
2     // on se connecte
3     $imapResource = imap_open($name, $infos["user"], $infos["password"]);
4     if (!$imapResource) {
5       // échec
6       print "La connexion au serveur [$name] a échoué : " . imap_last_error() . "\n";
7       exit;
8     }
9     // Connexion établie
10    print "Connexion établie avec le serveur [$name].\n";
11    // total des messages dans la boîte à lettres
12    $nbmsg = imap_num_msg($imapResource);
13    print "Il y a [$nbmsg] messages dans la boîte à lettres [$name]\n";
14    // messages non lus dans la boîte aux lettres courante
15    if ($nbmsg > 0) {
16      print "Récupération de la liste des messages non lus de la boîte à lettres [$name]\n";
17      $msgNumbers = imap_search($imapResource, 'UNSEEN');
18      if ($msgNumbers === FALSE) {
19        print "Il n'y a pas de nouveaux messages dans la boîte à lettres [$name]\n";
20      } else {
21        // on parcourt la liste des messages non lus
22        foreach ($msgNumbers as $msgNumber) {
23          print "---message n° [$msgNumber]\n";
24          // on récupère le corps du message n° $msgNumber
25          getMailBody($imapResource, $msgNumber, $infos);
26          // si le protocole est POP3, on supprime le message après l'avoir récupéré
27          $pop3 = $infos["pop3"];
28          if ($pop3 !== NULL) {
29            // on marque le message comme "à supprimer"
30            imap_delete($imapResource, $msgNumber);
31          }
32        }
33        // fin de la lecture des messages non lus
34        if ($pop3 !== NULL) {
35          // on supprime les messages marqués comme "à supprimer"
36          imap_expunge($imapResource);
37        }
38      }
39    }
40    // fermeture de la connexion
41    $imapClose = imap_close($imapResource);
42    if (!$imapClose) {
43      // échec
44      print "La fermeture de la connexion a échoué : " . imap_last_error() . "\n";
45    } else {
46      // réussite
47      print "Fermeture de la connexion réussie.\n";
48    }
49  }
```

Commentaires

Le code de la fonction **[readmailbox]** est le même que dans les scripts précédents.

La fonction **[getMailBody]** (ligne 25) qui analyse le corps d'un message (contenu + attachements) est la suivante :

```
1   // analyse du corps du message
2   function getMailBody($imapResource, int $msgNumber, array $infos): void {
3     // on récupère le texte entier du message
4     $text = imap_fetchbody($imapResource, $msgNumber, "");
5     if ($text === FALSE) {
6       print "Le corps du message [$msgNumber] n'a pu être récupéré";
7       return;
8     }
9     // on crée un parseur qui va analyser le texte du message
10    $parser = (new PhpMimeMailParser\Parser())->setText($text);
11    // on récupère les différentes parties du message
12    $outputDir = $infos["output-dir"] . "/message-$msgNumber";
13    getParts($parser, $msgNumber, $outputDir);
```

```
14  }
```

Commentaires

- ligne 2 : la fonction **[getMailBody]** accepte trois paramètres :
  - **[$imapResource]** : la ressource IMAP à laquelle on est connecté ;
  - **[$msgNumber]** : le n° du message (dans la boîte à lettres) à exploiter ;
  - **[$infos]** : informations diverses sur la boîte à lettres exploitée ;
- ligne 4 : on récupère la totalité du message n° **[$msgNumber]** ;
- lignes 5-8 : cas où le contenu du message n'a pu être récupéré ;
- ligne 10 : on commence à utiliser la bibliothèque **[php-mime-mail-parser]**. L'objet **[$parser]** va être chargé d'analyser le texte du message ;
- ligne 12 : **[$outputDir]** va être le dossier dans lequel vont être sauvegardés les contenus texte et les attachements du message n° **[$msgNumber]** ;
- ligne 13 : on demande à la fonction **[getParts]** de trouver les différentes parties (contenus texte et attachements) du message n° **[$msgNumber]** et de les sauvegarder dans le dossier **[$outputDir]** ;

La fonction **[getParts]** est la suivante :

```
1   // récupération des différentes parties d'un message
2   function getParts(PhpMimeMailParser\Parser $parser, int $msgNumber, string $outputDir): void {
3     // on crée le dossier de sauvegarde du message si besoin est
4     if (!file_exists($outputDir)) {
5       if (!mkdir($outputDir)) {
6         print "Le dossier [$outputDir] n'a pu être créé\n";
7         return;
8       }
9     }
10    // on récupère les entêtes du message
11    $arrayHeaders = $parser->getHeaders();
12    // on sauvegarde les messages texte
13    $parts = $parser->getInlineParts("text");
14    for ($i = 1; $i <= count($parts); $i++) {
15      print "-- Sauvegarde d'un message de type [text/plain]\n";
16      saveMessage($parts[$i - 1], 0, $arrayHeaders, "$outputDir/message_$i.txt");
17    }
18    // on sauvegarde les messages html
19    $parts = $parser->getInlineParts("html");
20    for ($i = 1; $i <= count($parts); $i++) {
21      print "-- Sauvegarde d'un message de type [text/html]\n";
22      saveMessage($parts[$i - 1], 1, $arrayHeaders, "$outputDir/message_$i.html");
23    }
24    // on récupère les attachements du message
25    $attachments = $parser->getAttachments();
26    // n° de l'attachement
27    $iAttachment = 0;
28    // on parcourt la liste des attachements
29    foreach ($attachments as $attachment) {
30      // type d'attachement
31      $fileType = $attachment->getContentType();
32      print "-- Sauvegarde d'un attachement de type [$fileType] dans le fichier
    [$outputDir/{$attachment->getFilename()}]\n";
33      // on sauvegarde l'attachement
34      try {
35        $attachment->save($outputDir, PhpMimeMailParser\Parser::ATTACHMENT_DUPLICATE_SUFFIX);
36      } catch (Exception $e) {
37        print "L'attachement n'a pu être sauvegardé : " . $e->getMessage() . "\n";
38      }
39      // cas particulier du type message/rfc822
40      if ($fileType === "message/rfc822") {
41        // l'attachement est lui-même un message - on va le parser lui aussi
42        // on change de répertoire de sauvegarde
43        $iAttachment++;
44        $outputDir = $outputDir . "/rfc822-$iAttachment";
45        // on change le contenu à parser
46        $parser->setText($attachment->getContent());
47        // on analyse le message de façon récursive
48        getParts($parser, $msgNumber, $outputDir);
49      }
50    }
51  }
```

Commentaires

- ligne 2 : la fonction **[getParts]** admet trois paramètres :
  - un parseur **[$parser]** à qui on a transmis le texte total du message à analyser ;
  - **[$msgNumber]** est le n° du message en cours d'analyse ;
  - **[$outputDir]** est le dossier dans lequel les contenus et attachements du message doivent être sauvegardés ;
- lignes 4-9 : création du dossier **[$outputDir]** ;
- ligne 11 : on récupère les entêtes du message en cours d'analyse (from, to, subject…) ;
- ligne 13 : on récupère les parties du mail ayant le type **[text/plain]**. On récupère un tableau ;
- lignes 14-17 : on sauvegarde tous les éléments du tableau récupéré, en donnant à chacun un nom de fichier différent ;
- ligne 19 : on récupère les parties du mail ayant le type **[text/html]**. On récupère un tableau ;
- lignes 20-23 : on sauvegarde tous les éléments du tableau récupéré, en donnant à chacun un nom de fichier différent ;
- ligne 25 : on récupère la liste des attachements du message analysé ;
- ligne 29 : on parcourt cette liste ;
- ligne 24 : on récupère le type de l'attachement (attribut *Content-Type*) ;
- lignes 34-38 : sauvegarde de l'attachement dans le dossier **[$outputDir]**. Le second paramètre **[PhpMimeMailParser\Parser::ATTACHMENT_DUPLICATE_SUFFIX]** est une stratégie de nommage des pièces attachées. Si **[$attachment→getFilename()]** vaut X et que le fichier X existe déjà, alors la bibliothèque **[php-mime-mail-parser]** essaie les noms **[X_1]**, **[X_2]**, etc. jusqu'à trouver un nom de fichier qui n'existe pas ;
- ligne 40 : on regarde si la pièce attachée est un mail ;
- lignes 41-48 : si c'est le cas, alors ce mail est analysé à son tour pour en extraire contenus et attachements ;
- ligne 44 : si **[$outputDir]** vaut X et que parmi les attachements du message analysé il y a deux mails, alors le 1er sera sauvegardé dans le dossier **[$outputDir/rfc822-1]** et le second dans le dossier **[$outputDir/rfc822-2]** ;
- ligne 46 : le contenu du mail attaché devient le nouveau texte à parser ;
- ligne 48 : on appelle la fonction **[getParts]** de façon récursive pour analyser le nouveau texte ;

La fonction **[saveMessage]** sauve les contenus texte du message à analyser :

```
1   // sauvegarde d'un message texte
2   function saveMessage(string $text, int $type, array $arrayHeaders, string $filename): void {
3     // contenu à sauvegarder
4     $contents = "";
5     // ajout des entêtes
6     switch ($type) {
7       case 0:
8         // text/plain
9         foreach ($arrayHeaders as $key => $value) {
10          $contents .= "$key: $value\n";
11        }
12        $contents .= "\n";
13        break;
14      case 1:
15        // text/HTML
16        foreach ($arrayHeaders as $key => $value) {
17          $contents .= "$key: $value<br/>\n";
18        }
19        $contents .= "<br/>\n";
20    }
21    // ajout du texte du message
22    $contents .= $text;
23    // sauvegarde du tout
24    if (!file_put_contents($filename, $contents)) {
25      // échec
26      print "Le message n'a pu être sauvegardé dans le fichier [$filename]\n";
27    } else {
28      // réussite
29      print "Le message a été sauvegardé dans le fichier [$filename]\n";
30    }
31  }
```

Commentaires

- la fonction **[saveMessage]** admet les paramètres suivants :
  - **[$text]** : le texte à sauvegarder ;
  - **[$type]** : le type du texte (0 : text/plain, 1 : text/HTML) ;
  - **[$arrayHeaders]** : les entêtes du message analysé ;
  - **[$filename]** : le nom du fichier dans lequel doit être sauvegardé **[$text]** ;
- ligne 4 : **[$contents]** va représenter la totalité du texte à sauvegarder ;
- lignes 6-20 : on sauvegardera d'abord tous les entêtes du message (from, to, subject…) ;

- lignes 16-19 : dans le cas d'un texte HTML, on termine chaque ligne par la balise
 pour que chaque entête apparaisse seule sur sa ligne dans un navigateur ;
- ligne 22 : on ajoute aux entêtes le texte du message à sauvegarder ;
- lignes 24-30 : l'ensemble est sauvegardé dans le fichier **[$filename]** ;

L'utilisation de la bibliothèque **[php-mime-mail-parser]** facilite considérablement l'écriture du script de lecture de mails.

Le script **[smtp-02.php]** est utilisé pour envoyer un mail à l'utilisateur **[guest@localhost]** avec la configuration suivante :

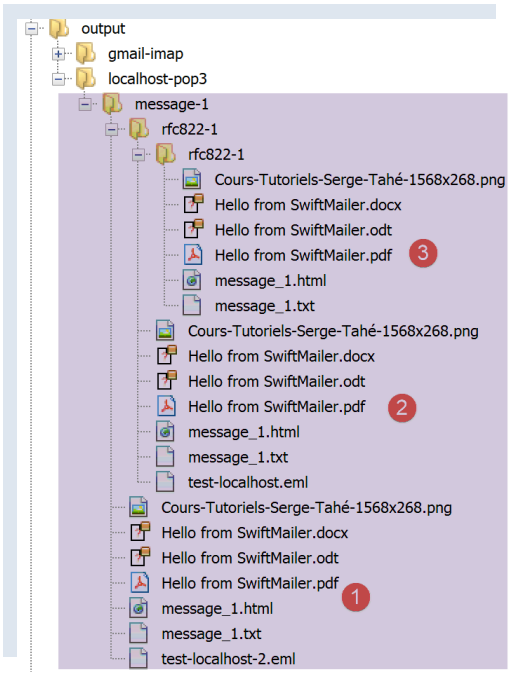
```
1.  {
2.    "mail to localhost via localhost": {
3.      "smtp-server": "localhost",
4.      "smtp-port": "25",
5.      "from": "guest@localhost",
6.      "to": "guest@localhost",
7.      "subject": "test-localhost",
8.      "message": "ligne 1\nligne 2\nligne 3",
9.      "tls": "FALSE",
10.     "attachments": [
11.       "/attachments/Hello from SwiftMailer.docx",
12.       "/attachments/Hello from SwiftMailer.pdf",
13.       "/attachments/Hello from SwiftMailer.odt",
14.       "/attachments/Cours-Tutoriels-Serge-Tahé-1568x268.png",
15.       "/attachments/test-localhost-2.eml"
16.     ]
17.   }
18. }
```

- lignes 11-15 : il y a cinq attachements ;
- ligne 15 : **[test-localhost-2.eml]** est un mail structuré de la façon suivante :
  - **[test-localhost-2.eml]** contient 4 attachements (les mêmes qu'aux lignes 11-14) et un mail attaché ;
  - le mail attaché à **[test-localhost-2.eml]** contient 4 attachements (les mêmes qu'aux lignes 11-14) ;

Le script **[imap-03.php]** est utilisé pour lire la boîte à lettres de l'utilisateur **[guest@localhost]** avec la configuration suivante :

```
1.  {
2.    "{localhost:110/pop3}": {
3.      "imap-server": "localhost",
4.      "imap-port": "110",
5.      "user": "guest@localhost",
6.      "password": "guest",
7.      "pop3": "TRUE",
8.      "output-dir": "output/localhost-pop3"
9.    }
10. }
```

Après exécution, l'arborescence du dossier **[output/localhost-pop3]** est devenue la suivante :



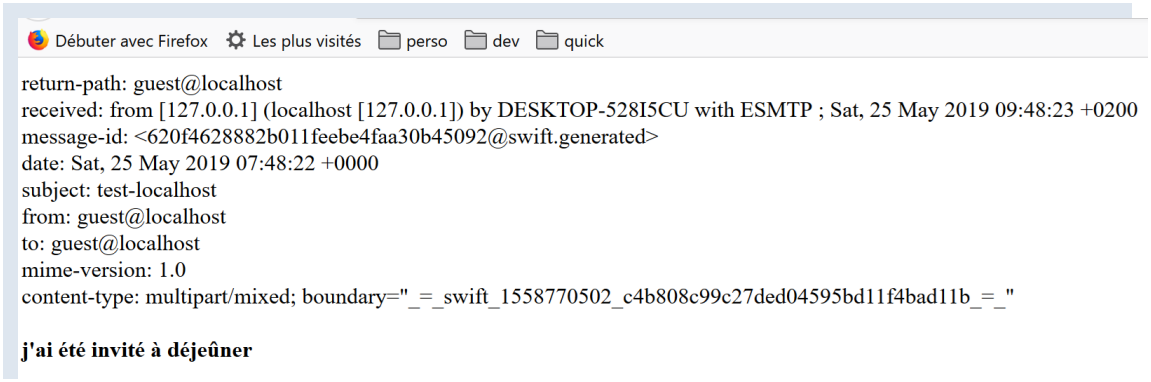
- en **[1]**, les 5 attachements du mail reçu par **[guest@localhost]** ;
- en **[2]**, les 5 attachements du mail **[test-localhost-2.eml]** de **[1]** ;
- en **[3]**, les 4 attachements du mails **[test-localhost.eml]** de **[2]** ;

Les affichages console sont les suivants :

```
1   ------------Lecture de la boîte à lettres [{localhost:110/pop3}]
2   Connexion établie avec le serveur [{localhost:110/pop3}].
3   Il y a [1] messages dans la boîte à lettres [{localhost:110/pop3}]
4   Récupération de la liste des messages non lus de la boîte à lettres [{localhost:110/pop3}]
5   ---message n° [1]
6   -- Sauvegarde d'un message de type [text/plain]
7   Le message a été sauvegardé dans le fichier [output/localhost-pop3/message-1/message_1.txt]
8   -- Sauvegarde d'un message de type [text/html]
9   Le message a été sauvegardé dans le fichier [output/localhost-pop3/message-1/message_1.html]
10  -- Sauvegarde d'un attachement de type [application/vnd.openxmlformats-
    officedocument.wordprocessingml.document] dans le fichier [output/localhost-pop3/message-1/Hello from
    SwiftMailer.docx]
11  -- Sauvegarde d'un attachement de type [application/pdf] dans le fichier [output/localhost-pop3/message-
    1/Hello from SwiftMailer.pdf]
12  -- Sauvegarde d'un attachement de type [application/vnd.oasis.opendocument.text] dans le fichier
    [output/localhost-pop3/message-1/Hello from SwiftMailer.odt]
13  -- Sauvegarde d'un attachement de type [image/png] dans le fichier [output/localhost-pop3/message-1/Cours-
    Tutoriels-Serge-Tahé-1568x268.png]
14  -- Sauvegarde d'un attachement de type [message/rfc822] dans le fichier [output/localhost-pop3/message-
    1/test-localhost-2.eml]
15  -- Sauvegarde d'un message de type [text/plain]
16  Le message a été sauvegardé dans le fichier [output/localhost-pop3/message-1/rfc822-1/message_1.txt]
17  -- Sauvegarde d'un message de type [text/html]
18  Le message a été sauvegardé dans le fichier [output/localhost-pop3/message-1/rfc822-1/message_1.html]
19  -- Sauvegarde d'un attachement de type [application/vnd.openxmlformats-
    officedocument.wordprocessingml.document] dans le fichier [output/localhost-pop3/message-1/rfc822-1/Hello
    from SwiftMailer.docx]
20  -- Sauvegarde d'un attachement de type [application/pdf] dans le fichier [output/localhost-pop3/message-
    1/rfc822-1/Hello from SwiftMailer.pdf]
21  -- Sauvegarde d'un attachement de type [application/vnd.oasis.opendocument.text] dans le fichier
    [output/localhost-pop3/message-1/rfc822-1/Hello from SwiftMailer.odt]
22  -- Sauvegarde d'un attachement de type [image/png] dans le fichier [output/localhost-pop3/message-1/rfc822-
    1/Cours-Tutoriels-Serge-Tahé-1568x268.png]
23  -- Sauvegarde d'un attachement de type [message/rfc822] dans le fichier [output/localhost-pop3/message-
    1/rfc822-1/test-localhost.eml]
24  -- Sauvegarde d'un message de type [text/plain]
25  Le message a été sauvegardé dans le fichier [output/localhost-pop3/message-1/rfc822-1/rfc822-
    1/message_1.txt]
26  -- Sauvegarde d'un message de type [text/html]
```

```
27  Le message a été sauvegardé dans le fichier [output/localhost-pop3/message-1/rfc822-1/rfc822-
    1/message_1.html]
28  -- Sauvegarde d'un attachement de type [application/vnd.openxmlformats-
    officedocument.wordprocessingml.document] dans le fichier [output/localhost-pop3/message-1/rfc822-1/rfc822-
    1/Hello from SwiftMailer.docx]
29  -- Sauvegarde d'un attachement de type [application/pdf] dans le fichier [output/localhost-pop3/message-
    1/rfc822-1/rfc822-1/Hello from SwiftMailer.pdf]
30  -- Sauvegarde d'un attachement de type [application/vnd.oasis.opendocument.text] dans le fichier
    [output/localhost-pop3/message-1/rfc822-1/rfc822-1/Hello from SwiftMailer.odt]
31  -- Sauvegarde d'un attachement de type [image/png] dans le fichier [output/localhost-pop3/message-1/rfc822-
    1/rfc822-1/Cours-Tutoriels-Serge-Tahé-1568x268.png]
32  Fermeture de la connexion réussie.
```

Si on visualise **[message_1.HTML]** de **[3]** dans un navigateur, on obtient la chose suivante :



1.17 Services web

Note : par *service web*, on entend ici tout application web délivrant des données brutes consommées par un client, un script console dans les exemples qui vont suivre. On ne s'intéresse pas à une technologie particulière, REST (**RE**presentational **S**tate **T**ransfer) ou SOAP (**S**imple **O**bject **A**ccess **P**rotocol) par exemple, qui délivrent des données plus ou moins brutes dans un format bien défini. REST délivre du jSON alors que pour SOAP c'est du XML. Chacune de ces technologies décrit précisément la façon dont le client doit interroger le serveur et la forme que doit prendre la réponse de celui-ci. Dans ce cours, on sera beaucoup plus souple quant à la nature de la requête client et celle de la réponse du serveur. Cependant, les scripts écrits et les outils utilisés sont proches de ceux de la technologie REST.

1.17.1 Introduction

Les programmes PHP pouvant être exécutés par un serveur WEB, un tel programme devient un programme serveur pouvant servir plusieurs clients. Du point de vue du client, appeler un service web revient à demander l'URL de ce service. Le client peut être écrit avec n'importe quel langage, notamment en PHP. Dans ce dernier cas, on utilise alors les fonctions réseau que nous venons de voir. Il nous faut par ailleurs savoir "converser" avec un service web, c'est-à-dire comprendre le protocole *http* de communication entre un serveur WEB et ses clients. C'était le but du paragraphe lien.

Le client web décrit au paragraphe lien nous a permis de découvrir une partie du protocole HTTP.

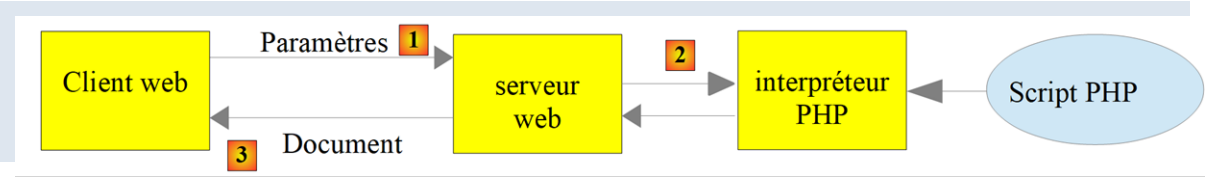
Dans leur version la plus simple, les échanges client / serveur sont les suivants :

- le client ouvre une connexion avec le port 80 du serveur web ;
- il fait une requête concernant un document ;
- le serveur web envoie le document demandé et ferme la connexion ;
- le client ferme à son tour la connexion ;

Le document peut être de nature diverse : un texte au format HTML, une image, une vidéo… Ce peut être un document existant (document statique) ou bien un document généré à la volée par un script (document dynamique). Dans ce dernier cas, on parle de

programmation web. Le script de génération dynamique de documents peut être écrit dans divers langages : PHP, Python, Perl, Java, Ruby, C#, VB.net…

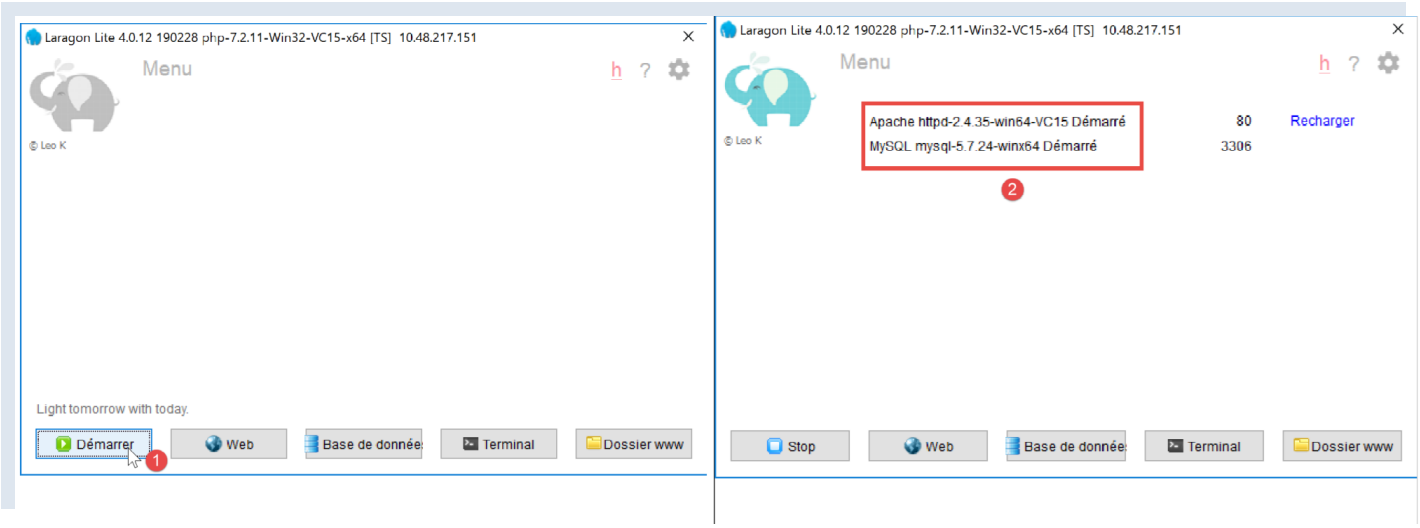
Dans la suite, nous allons utiliser des scripts PHP pour générer dynamiquement des documents texte.



- en **[1]**, le client ouvre une connexion avec le serveur, demande un script PHP, envoie ou non des paramètres à destination de ce script ;
- en **[2]**, le serveur web fait exécuter le script PHP par l'interpréteur PHP. Le script génère un document qui est envoyé au client **[3] ;**
- le serveur clôt la connexion. Le client en fait autant ;

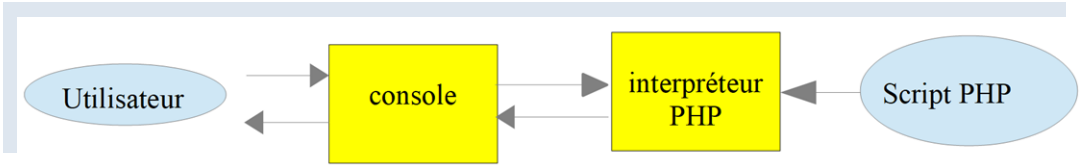
Le serveur web peut traiter plusieurs clients à la fois.

Avec le paquetage logiciel **[Laragon]**, le serveur web est un serveur Apache, un serveur open source de l'Apache Foundation (http://www.apache.org/). Dans les applications qui suivent, **[Laragon]** doit être lancé :



Cela lance le serveur web Apache ainsi que le SGBD MySQL.

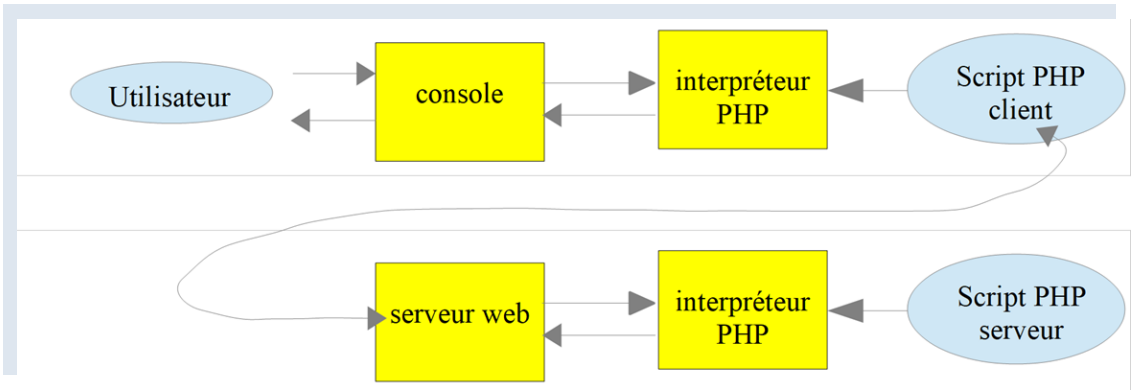
Les scripts exécutés par le serveur web seront écrits avec l'outil Netbeans. Nous avons écrit jusqu'à maintenant des scripts PHP exécutés dans un contexte console :



L'utilisateur utilise la console pour demander l'exécution d'un script PHP et en recevoir les résultats.

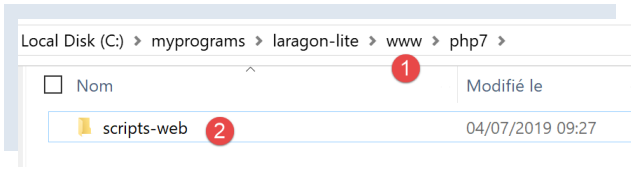
Dans les applications client /serveur qui vont suivre :

- le script du client est exécuté dans un contexte console ;
- le script du serveur est exécuté dans un contexte web ;



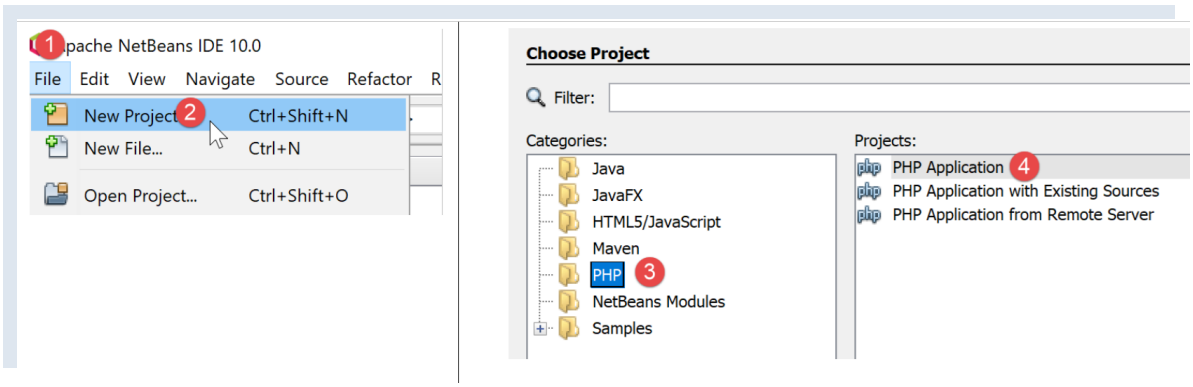
Le script PHP du serveur ne peut pas être n'importe où dans le système de fichiers. En effet, le serveur web cherche dans des endroits précisés par configuration, les documents statiques et dynamiques qu'on lui demande. La configuration par défaut de Laragon fait que les documents sont cherchés dans le dossier *<Laragon>/www* où *<Laragon>* est le dossier d'installation de Laragon. Ainsi si un client web demande un document D avec l'URL **[http://localhost/D]**, le serveur web lui servira le document D de chemin **[<Laragon>/www/D]**.

Dans les exemples qui suivent, nous mettrons les scripts serveur dans le dossier **[www/php7/scripts-web]**. Si un script serveur s'appelle *S.php*, il sera demandé au serveur web avec l'URL **[http://localhost/php7/scripts-web/S.php]**. Le document **[<Laragon>/www/php7/scripts-web/S.php]** lui sera alors servi.

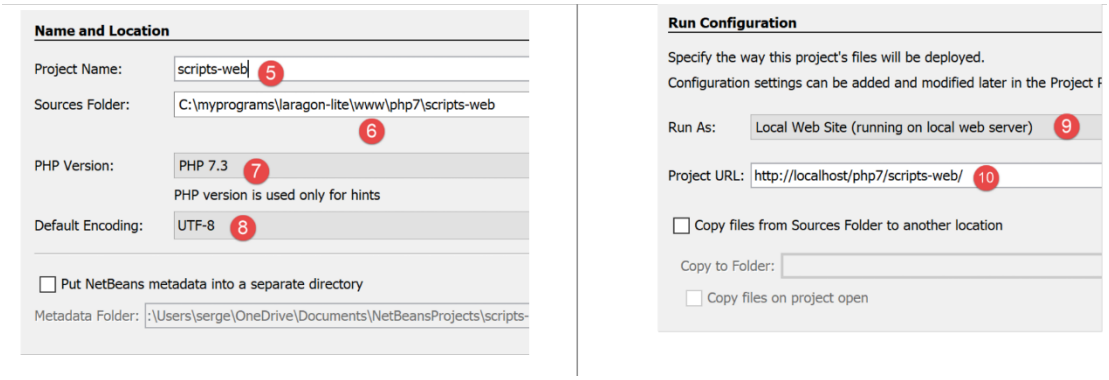


- en **[1]**, le dossier **[<laragon>/www]** ;
- en **[2]**, le dossier **[php7/scripts-web]** ;

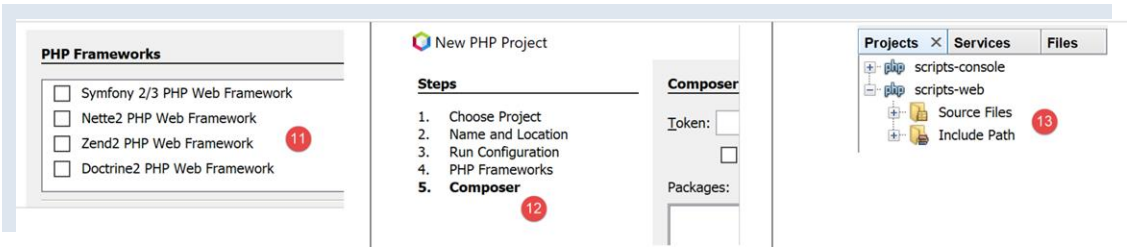
Pour créer des scripts serveur avec Netbeans, nous procéderons de la façon suivante :



- en **[1-2]**, nous créons un nouveau projet
- en **[3-4]**, nous prenons la catégorie **[PHP]** et le projet **[PHP Application]**



- en **[5]**, le nom du projet ;
- en **[6]**, le dossier du projet dans le système de fichiers. Notez que celui-ci est dans le dossier **[<laragon>/www]** où il doit être ;
- en **[7-8]**, acceptez les valeurs par défaut proposées ;
- en **[9-10]**, acceptez les valeurs par défaut proposées. En **[10]**, notez que l'URL des scripts que nous placerons dans ce projet commencera par le chemin **[http://localhost/php7/scripts-web/]** ;

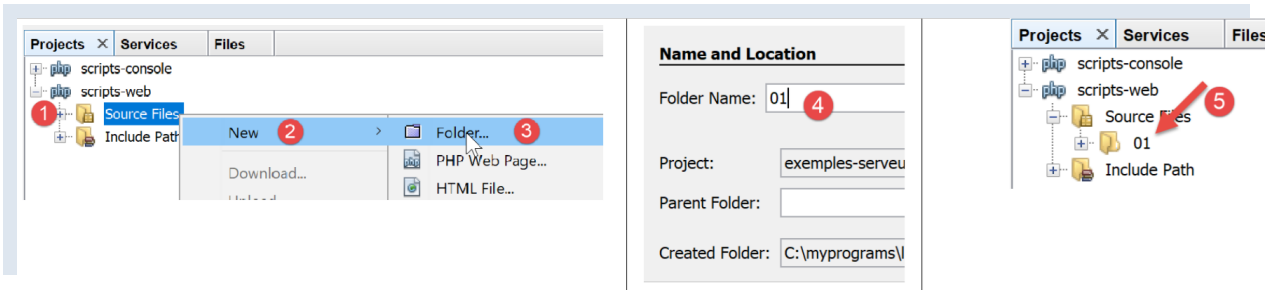


- en **[11]**, des frameworks web écrits en PHP vous sont proposés. Ces frameworks sont indispensables dès que l'application web prend un peu d'ampleur ;
- en **[12]**, on peut ajouter des bibliothèques PHP à l'aide de l'outil **[Composer]**. Nous avons utilisé cet outil à deux reprises dans une fenêtre **[Terminal]** de Laragon :
  - pour installer la bibliothèque **[SwiftMailer]** qui permet d'envoyer des mails ;
  - pour installer la bibliothèque **[php-mime-mail-parser]** qui permet de lire des mails ;
- en **[13]**, une fois l'assistant de création du projet validé, celui-ci apparaît en **[13]** dans l'onglet des projets ;

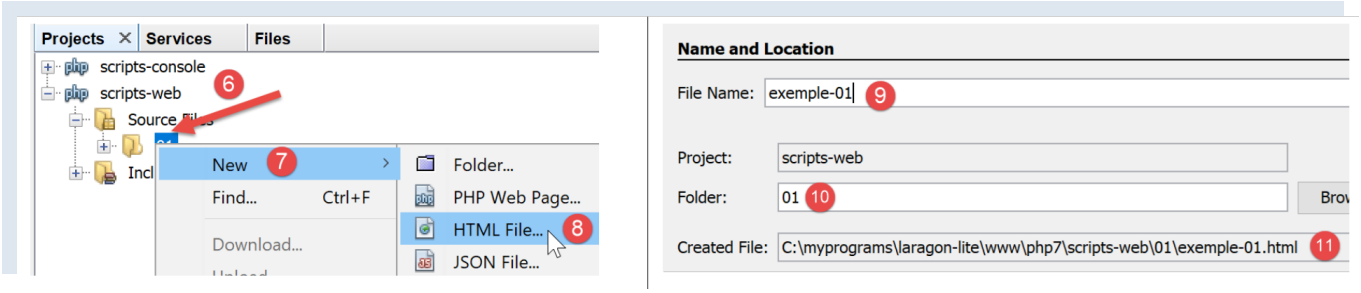
1.17.2 Ecriture d'une page statique

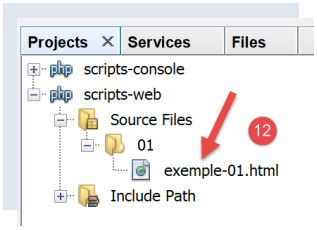
Note : Pour la suite, il faut que **[Laragon]** soit lancé.

Nous allons montrer comment créer une page statique HTML (HyperText Markup Language) à l'aide de Netbeans :



- en **[1-5]**, nous créons un dossier nommé **[01]** ;





- en **[6-12]**, nous créons un fichier HTML **[exemple-01.html]** ;

Le fichier **[exemple-01.html]** est généré prérempli de la façon suivante (mai 2019) :

```
1.  <!DOCTYPE html>
2.  <!--
3.  To change this license header, choose License Headers in Project Properties.
4.  To change this template file, choose Tools | Templates
5.  and open the template in the editor.
6.  -->
7.  <html>
8.    <head>
9.      <title>TODO supply a title</title>
10.     <meta charset="UTF-8" />
11.     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
12.   </head>
13.
14.   <body>
15.     <div>TODO write content</div>
16.   </body>
17. </html>
```

Faisons évoluer son contenu de la façon suivante :

```
1.  <!DOCTYPE html>
2.  <html>
3.    <head>
4.      <title>PHP7 par l'exemple</title>
5.      <meta charset="UTF-8" />
6.      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7.    </head>
8.
9.    <body>
10.     <div><b>Ceci est un exemple de page statique</b></div>
11.   </body>
12. </html>
```

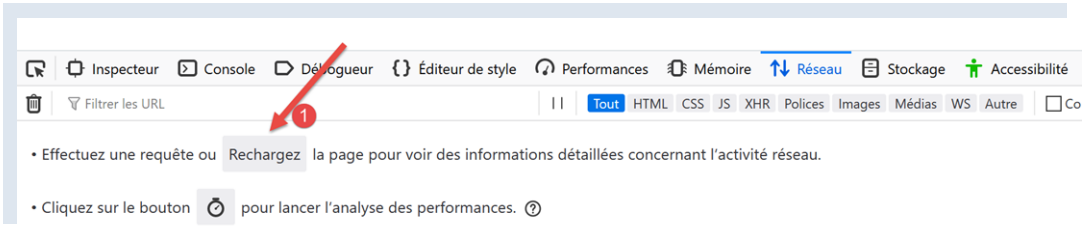
Nous avons changé le titre de la page (ligne 4) ainsi que son contenu (ligne 9).

Maintenant faisons afficher cette page HTML par le serveur Apache de Laragon :

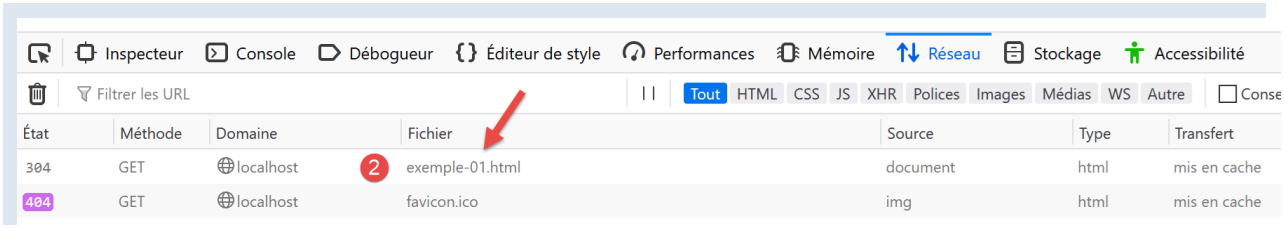
- en **[1-2]**, nous faisons afficher la page par le serveur Apache de Laragon ;
- en **[3]**, l'URL de la page affichée ;
- en **[4]**, le titre que nous avons modifié ;
- en **[5]**, le contenu que nous avons modifié ;

La page affichée est une page statique : on peut la charger autant de fois que l'on veut dans le navigateur (F5), c'est toujours le même contenu qui est affiché.

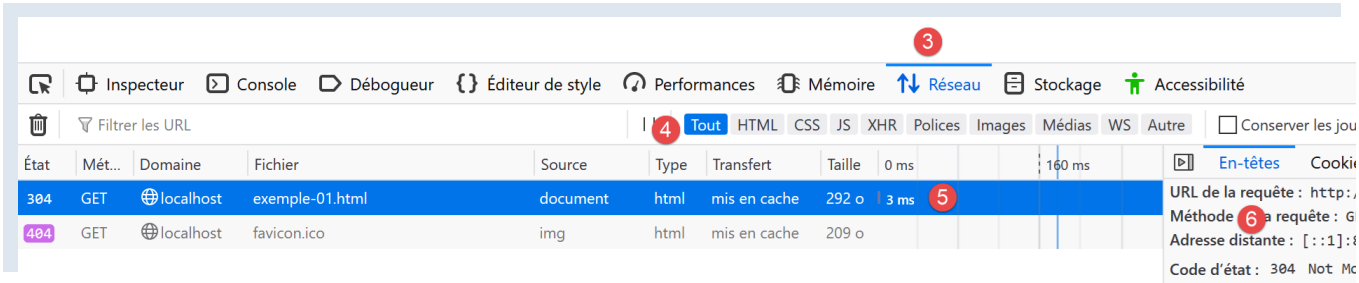
La plupart des navigateurs donnent accès aux données échangées entre le client et le serveur, celles qui ont été décrites au paragraphe lien. Avec le navigateur Firefox (mai 2019), il faut faire F12 pour avoir accès à ces données :



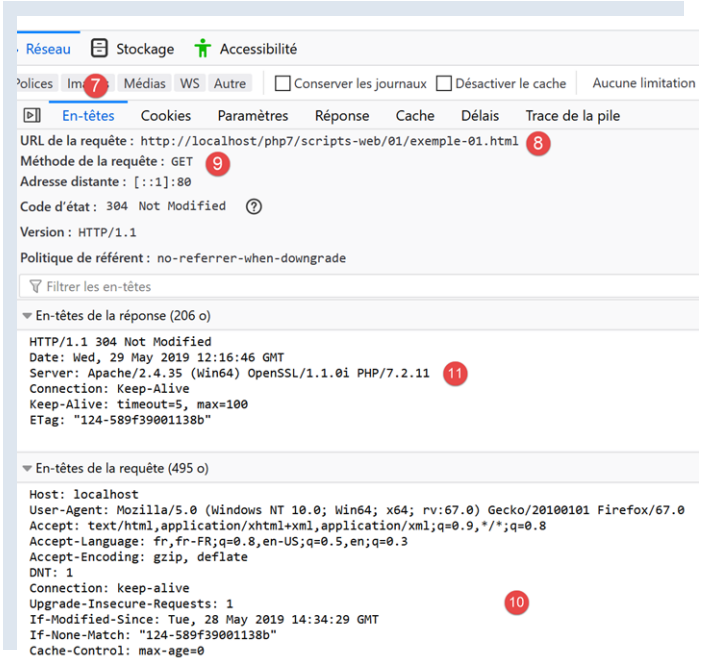
Comme indiqué en **[1]**, rechargeons la page (F5) :



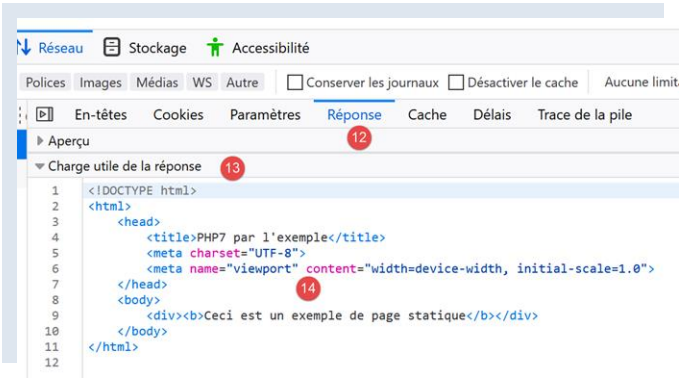
- en **[2]**, le document chargé par le navigateur : nous le sélectionnons ;



- en **[5]**, le document à analyser est sélectionné ;
- en **[3-4]**, nous demandons à voir les échanges client / serveur ;
- en **[6]**, ces échanges ;



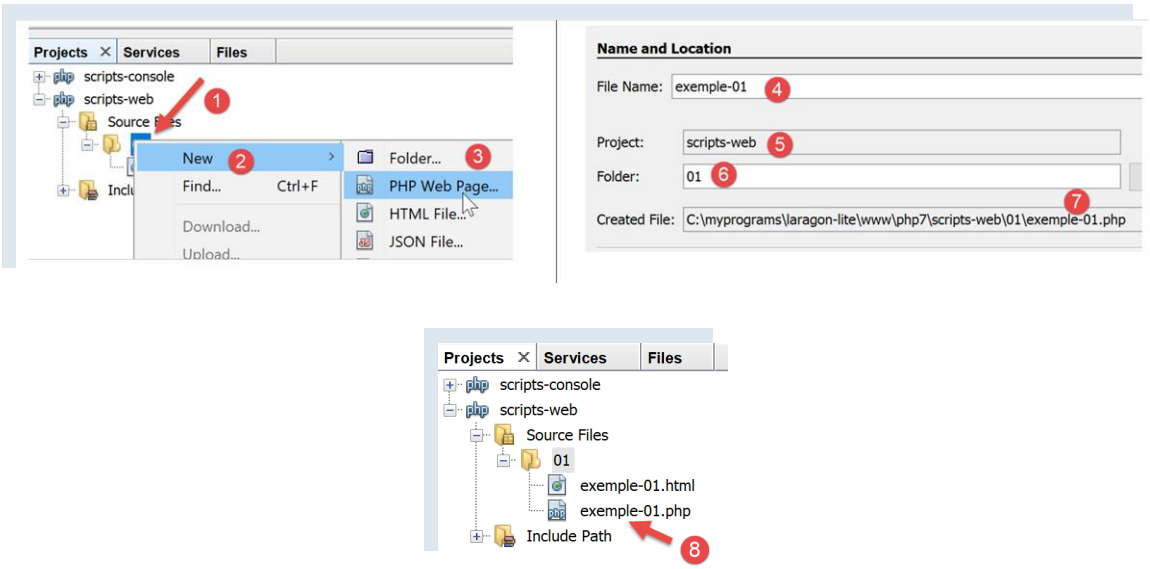
- en **[7]**, on sélectionne l'onglet des entêtes ;
- en **[8]**, l'URL demandée par le navigateur ;
- en **[9]**, la commande envoyée au serveur est **[GET http://localhost/php7/scripts-web/01/exemple-01.html HTTP/1.1]** ;
- en **[10]**, les entêtes HTTP envoyés ensuite par le navigateur (le client) ;
- en **[11]**, les entêtes HTTP de la réponse du serveur ;



- en **[12-14]**, la réponse du serveur envoyée après les entêtes HTTP ;
- en **[14]**, on voit que le navigateur client a reçu la page HTML que nous avons construite. Il a ensuite interprété ce code pour afficher la chose suivante :

1.17.3 Création d'une page dynamique en PHP

Nous écrivons maintenant une page dynamique en PHP :



- en **[1-8]**, nous créons une page **[exemple-01.php]** ;

Le fichier **[exemple-01.php]** est généré prérempli de la façon suivante (mai 2019) :

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
  <head>
    <meta charset="UTF-8" />
    <title></title>
  </head>

  <body>
    <?php
        // put your code here
        ?>
  </body>
</html>
```

Nous faisons évoluer le code ci-dessus de la façon suivante :

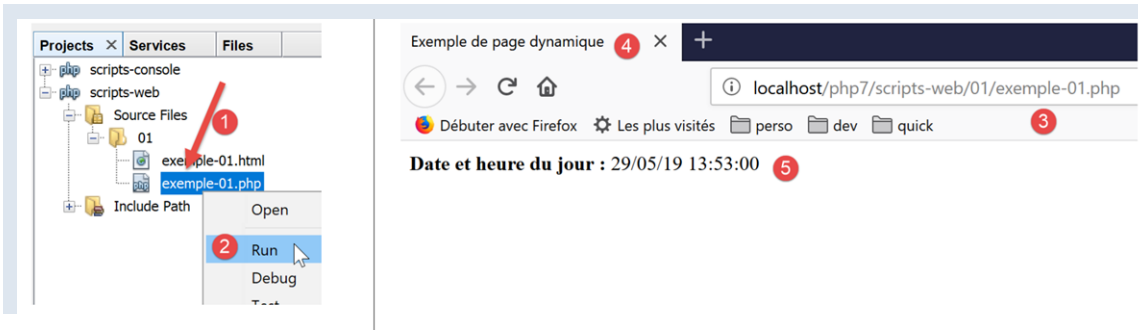
```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8">
  <title>Exemple de page dynamique</title>
</head>

<body>
  <?php
        // time : nb de millisecondes entre le moment présent et le 01/01/1970
        // format affichage date-heure
        // d : jour sur 2 chiffres
        // m : mois sur 2 chiffres
        // y : année sur 2 chiffres
        // H : heure 0,23
        // I : minutes
        // s: secondes
        print "<b>Date et heure du jour : </b>" . date("d/m/y H:i:s", time());
        ?>
</body>

</html>
```

Commentaires

- ligne 6 : nous avons changé le titre de la page ;
- ligne 19 : écrit la date et l'heure du moment présent ;

Basiquement, le script PHP ci-dessus écrit l'heure courante sur la console. Cependant, lorsqu'il est exécuté par un serveur web, le flux de sortie de l'instruction **[print]** qui est habituellement associé à la console d'exécution du script est redirigé ici vers la connexion qui lie le serveur à son client. Donc, dans un contexte web, le script ci-dessus envoie l'heure courante sous forme de texte au client, ici un navigateur.

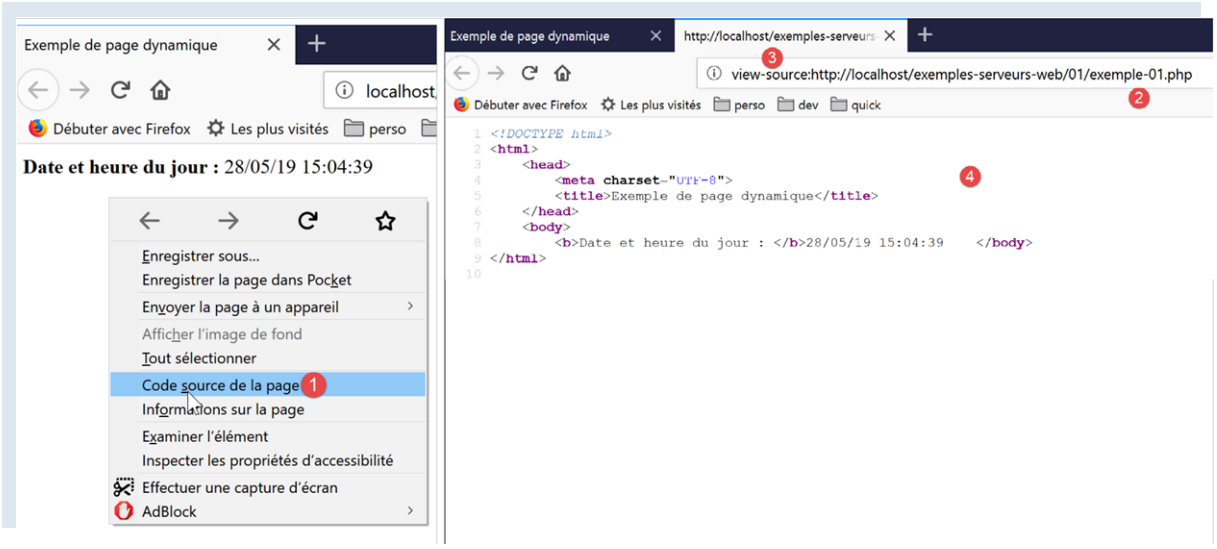
Exécutons le script **[exemple-01.php]** :



- en **[3]**, l'URL demandée au serveur web Apache ;
- en **[4]**, le titre de la page que nous avons changé ;
- en **[5]**, le contenu généré par l'instruction **[print]** ;

Nous avons ici une page **dynamique** car si on recharge plusieurs fois la page dans le navigateur (F5), son contenu change (l'heure change).

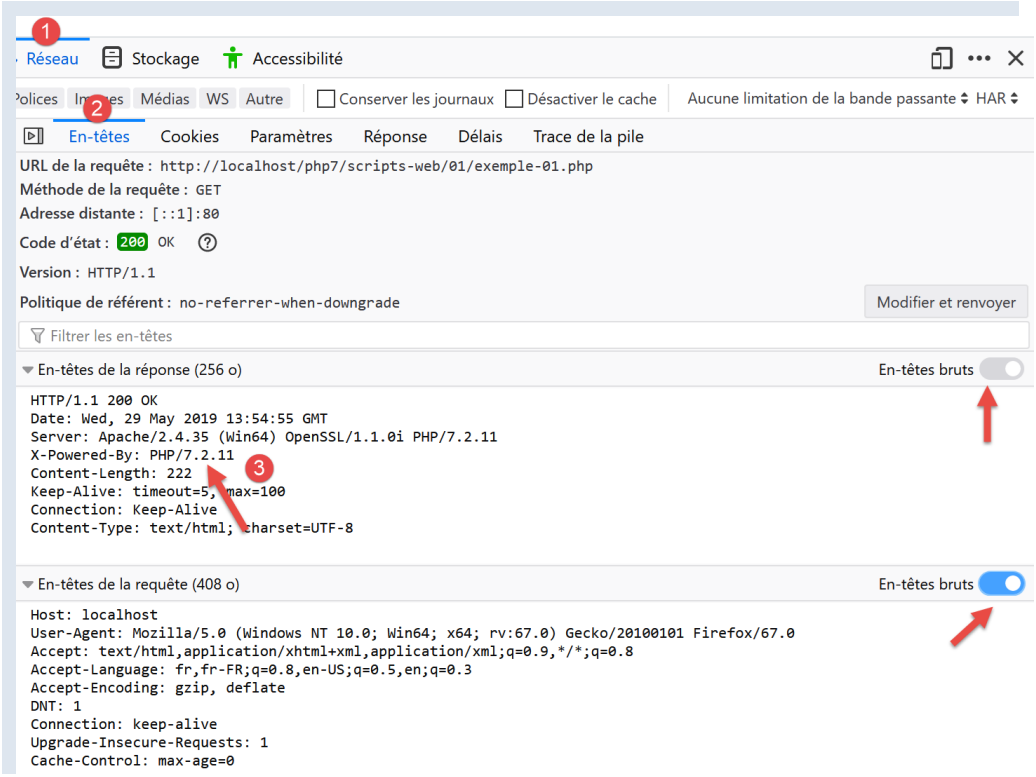
Le navigateur a reçu un flux HTML. Pour connaître celui-ci, il faut faire apparaître le code source de la page dans le navigateur :



- pour avoir le menu **[1]**, cliquer droit sur la page dans le navigateur ;
- en **[2]**, l'URL de la page **[exemple-01.php]** mais préfixée par **[view-source :]** **[3]** ;
- en **[4]**, le contenu HTML que le navigateur a affiché ;

Il faut donc se rappeler qu'un script PHP destiné à être exécuté par un serveur web doit produire un flux HTML.

Regardons (F12) maintenant les entêtes HTTP envoyés par le serveur au navigateur client :



- en **[3]**, un entête HTTP qui n'était pas présent lorsqu'on a demandé la page statique. Cet entête montre que la réponse du serveur a été générée par un script PHP ;

Nous avons vu que la réponse (le flux HTML ici) du serveur pouvait être générée par un script PHP. Le script peut également générer les entêtes HTTP et à peu près tous les éléments de la réponse du serveur.

1.17.4 Rudiments du langage HTML

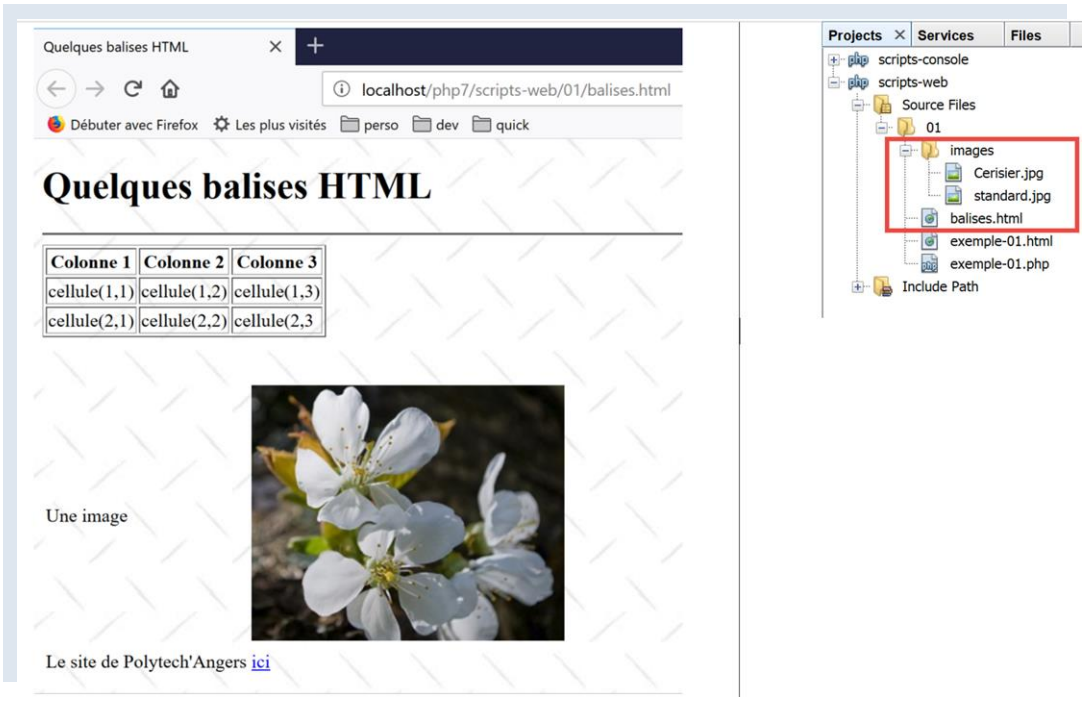
Ce chapitre ne va pas s'appesantir sur la programmation WEB en PHP. Une application web MVC est développée au paragraphe lien. Ce chapitre s'intéresse plutôt aux **services web** : des pages PHP qui délivrent, via un serveur web, des données à destination d'autres clients PHP. Néanmoins il nous a semblé utile de donner quelques rudiments de HTML au lecteur.

Un navigateur web peut afficher divers documents, le plus courant étant le document HTML (HyperText Markup Language). Celui-ci est un texte formaté avec des balises de la forme *<balise>texte</balise>*. Ainsi le texte *important* affichera le texte *important* en gras. Il existe des balises seules, telles que la balise *<hr/>* qui affiche une ligne horizontale. Nous ne passerons pas en revue les balises que l'on peut trouver dans un texte HTML. Il existe de nombreux logiciels WYSIWYG permettant de construire une page WEB sans écrire une ligne de code HTML. Ces outils génèrent automatiquement le code HTML d'une mise en page faite à l'aide de la souris et de contrôles prédéfinis. On peut ainsi insérer (avec la souris) dans la page un tableau puis consulter le code HTML généré par le logiciel pour découvrir les balises à utiliser pour définir un tableau dans une page WEB. Ce n'est pas plus compliqué que cela. Par ailleurs, la connaissance du langage HTML est indispensable puisque les applications web dynamiques doivent générer elles-mêmes le code HTML à envoyer aux clients WEB. Ce code est généré par programme et il faut bien sûr savoir ce qu'il faut générer pour que le client ait la page web qu'il désire.

Pour résumer, il n'est nul besoin de connaître la totalité du langage HTML pour démarrer la programmation web. Cependant cette connaissance est nécessaire et peut être acquise au travers de l'utilisation de logiciels WYSIWYG de construction de pages WEB tels que DreamWeaver et des dizaines d'autres. Une autre façon de découvrir les subtilités du langage HTML est de parcourir le web et d'afficher le code source des pages qui présentent des caractéristiques intéressantes et encore inconnues pour vous.

Considérons l'exemple suivant qui présente quelques éléments qu'on peut trouver dans un document WEB tels que :

- un tableau ;
- une image ;
- un lien.



Un document HTML a la forme générale suivante :

L'ensemble du document est encadré par les balises **<html>…</html>**. Il est formé de deux parties :

1. **<head>…</head>** : c'est la partie non affichable du document. Elle donne des renseignements au navigateur qui va afficher le document. On y trouve souvent la balise **<title>…</title>** qui fixe le texte qui sera affiché dans la barre de titre du navigateur. On peut y trouver d'autres balises notamment des balises définissant les mots clés du document, mot clés utilisés ensuite par les moteurs de recherche. On peut trouver également dans cette partie des scripts, écrits le plus souvent en javascript ou vbscript et qui seront exécutés par le navigateur.

2. **<body attributs>…</body>** : c'est la partie qui sera affichée par le navigateur. Les balises HTML contenues dans cette partie indiquent au navigateur la forme visuelle "souhaitée" pour le document. Chaque navigateur va interpréter ces balises à sa façon. Deux navigateurs peuvent alors visualiser différemment un même document web. C'est généralement l'un des casse-têtes des concepteurs web.

Le code HTML de notre document exemple est le suivant :

```
1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.      <title>Quelques balises HTML</title>
6.    </head>
7.
8.    <body style="background-image: url(images/standard.jpg)">
9.      <h1 style="text-align: left">Quelques balises HTML</h1>
10.     <hr />
11.
12.     <table border="1">
13.       <thead>
14.         <tr>
15.           <th>Colonne 1</th>
16.           <th>Colonne 2</th>
17.           <th>Colonne 3</th>
18.         </tr>
19.       </thead>
20.       <tbody>
21.         <tr>
22.           <td>cellule(1,1)</td>
23.           <td style="text-align: center">cellule(1,2)</td>
24.           <td>cellule(1,3)</td>
25.         </tr>
26.         <tr>
27.           <td>cellule(2,1)</td>
```

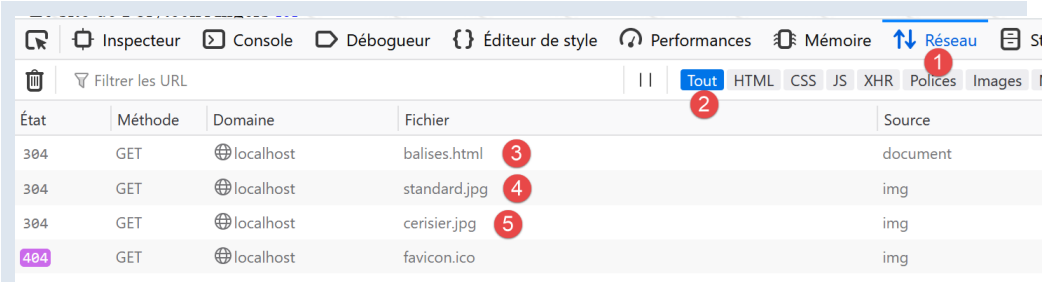
```
28.           <td>cellule(2,2)</td>
29.           <td>cellule(2,3</td>
30.         </tr>
31.       </tbody>
32.     </table>
33.     <br /><br />
34.     <table border="0">
35.       <tr>
36.         <td>Une image</td>
37.         <td>
38.           <img border="0" src="images/cerisier.jpg" />
39.         </td>
40.       </tr>
41.       <tr>
42.         <td>Le site de Polytech'Angers</td>
43.         <td><a href="http://www.polytech-angers.fr/fr/index.html">ici</a></td>
44.       </tr>
45.     </table>
46.   </body>
47. </html>
```

Elément	balises et exemples HTML
titre du document	`<title>Quelques balises HTML</title>` (ligne 5) le texte **[Quelques balises HTML]** apparaîtra dans la barre de titre du navigateur qui affichera le document
barre horizontale	`<hr />` : affiche un trait horizontal (ligne 10)
tableau	*<table attributs>….</table>* : pour définir le tableau (lignes 12, 32) *<thead>…</thead>* : pour définir les entêtes des colonnes (lignes 13, 19) *<tbody>…</tbody>* : pour définir le contenu du tableau (ligne 20, 31) *<tr attributs>…</tr>* : pour définir une ligne (lignes 21, 25) *<td attributs>…</td>* : pour définir une cellule (ligne 22) **exemples** : *<table border="1">…</table>* : l'attribut border définit l'épaisseur de la bordure du tableau `<td style="text-align: center;">cellule(1,2)</td>` (ligne 23) : définit une cellule dont le contenu sera **cellule(1,2)**. Ce contenu sera centré horizontalement (text-align :center).
image	`<img border="0" src="images/cerisier.jpg"/>` (ligne 38) : définit une image sans bordure (border=0") dont le fichier source est ***[images/cerisier.jpg]*** sur le serveur web (src="images/cerisier.jpg"). Ce lien se trouve sur un document web obtenu avec l'URL ***http://localhost/php7/scripts-web/01/****balises.html*. Aussi, le navigateur demandera-t-il l'URL ***http://localhost/php7/scripts-web/01/****images/cerisier.jpg* pour avoir l'image référencée ici.
lien	*ici* (ligne 42) : fait que le texte *ici* sert de lien vers l'URL `http://www.polytech-angers.fr/fr/index.html`.
fond de page	`<body style="background-image: url(images/standard.jpg)">` *(ligne 8)* : indique que l'image qui doit servir de fond de page se trouve à l'URL **[images/standard.jpg]** du serveur WEB. Dans le contexte de notre exemple, le navigateur demandera l'URL *http://localhost/php7/scripts-web/01/images/standard.jpg* pour obtenir cette image de fond.

On voit dans ce simple exemple que pour construire l'intéralité du document, le navigateur doit faire trois requêtes au serveur :

1. *http://localhost/php7/scripts-web/01/images/balises.html* pour avoir le source HTML du document
2. *http://localhost/php7/scripts-web/01/images/cerisier.jpg* pour avoir l'image *cerisier.jpg*
3. *http://localhost/php7/scripts-web/01/images/standard.jpg* pour obtenir l'image de fond *standard.jpg*

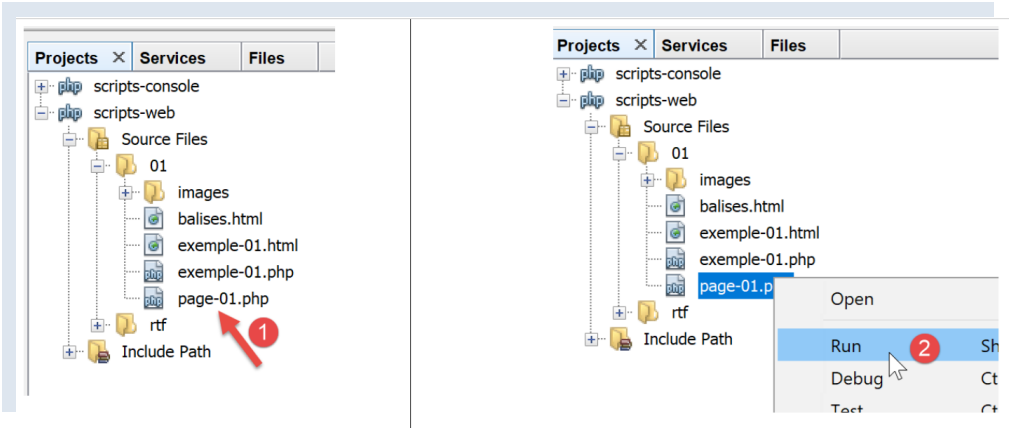
C'est ce que montrent les échanges réseau entre le client et le serveur (F12 dans le navigateur) :



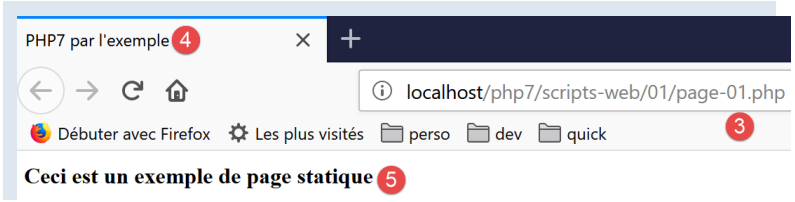
- en **[3-5]**, on voit les trois requêtes faites par le navigateur ;

1.17.5 Rendre dynamique une page statique

Montrons comment nous pouvons dynamiser la page HTML **[exemple-01.html]**. Copions le contenu

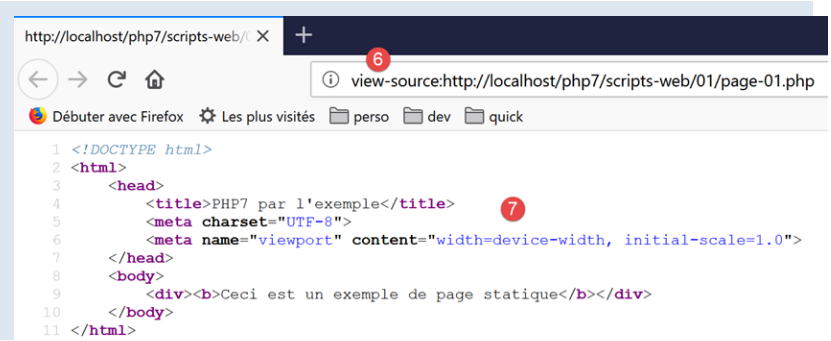


Nous avons recopié le contenu de **[exemple-01.html]** dans le fichier **[page-01.php]**. Si nous exécutons **[2]** ce script web, nous obtenons la chose suivante dans le navigateur :



- en **[3]**, l'URL demandée ;
- en **[4]**, le titre de la page ;
- en **[5]**, le contenu de la page ;

Si on fait afficher le code reçu par le navigateur, on trouve ceci :



- en **[7]**, on a le code HTML placé dans le script **[exemple-01.php] ;**

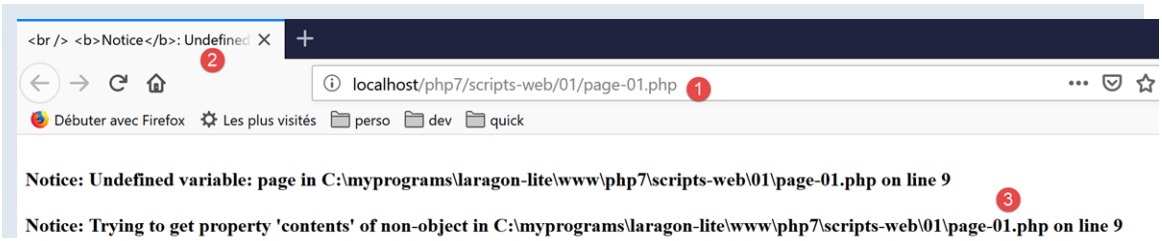
L'interpréteur PHP a interprété le script **[page-01.php]** et a produit le même flux HTML que la page statique **[exemple-01.html]**. Dans le script **[page-01.php]**, il n'y avait pas de PHP, uniquement du HTML. On apprend ainsi une chose : lorsque l'interpréteur PHP trouve du HTML dans un script PHP, il n'y touche pas et l'envoie tel quel au client.

Maintenant mettons quelques instructions PHP dans le script **[page-01.php]** pour que l'interpréteur PHP ait quelque chose à faire :

```
1.  <!DOCTYPE html>
2.  <html>
3.    <head>
4.      <title><?php print $page->title ?></title>
5.      <meta charset="UTF-8" />
6.      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7.    </head>
8.
9.    <body>
10.     <div>
11.       <b><?php print $page->contents ?></b>
12.     </div>
13.   </body>
14. </html>
```

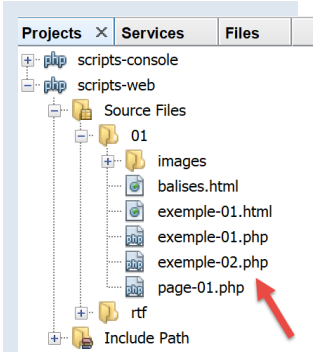
Lignes 4 et 11 on a mis du code PHP pour générer dynamiquement le titre et le contenu de la page. On fait ici l'hypothèse que la variable **[$page]** est un objet qui a les données à afficher.

Si nous exécutons ce nouveau code, on a le résultat suivant dans le navigateur :



- en **[1]**, l'URL demandée ;
- en **[2]**, le titre de la page n'a pu être affiché parce que la variable **[$page]** n'était pas définie ;
- en **[3]**, pareil pour le contenu ;

Maintenant, écrivons le script web **[exemple-02.php]** suivant :

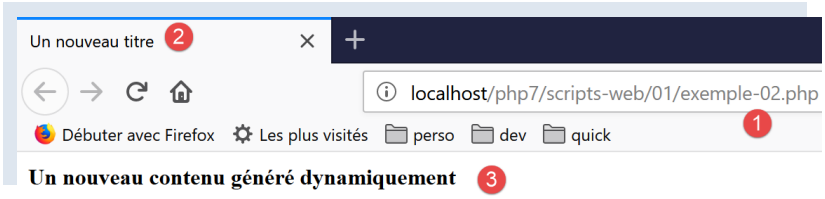


Le script **[exemple-02.php]** sera le suivant :

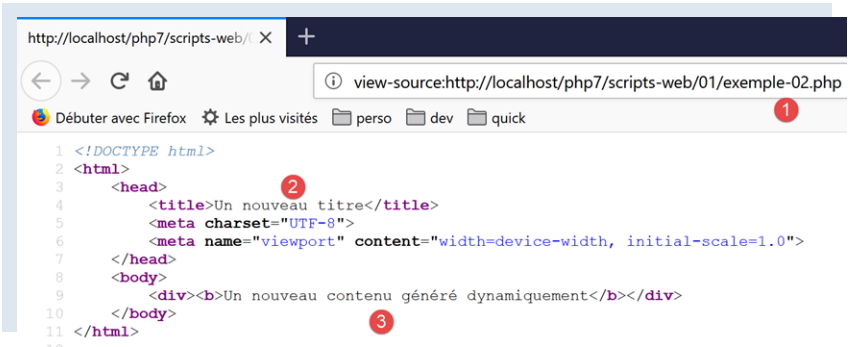
```
1   <?php
2
3   // on définit les éléments de la page à afficher
4   $page=new \stdclass();
5   $page->title="Un nouveau titre";
6   $page->contents="Un nouveau contenu généré dynamiquement";
7   // on fait afficher [page-01]
8   require_once "page-01.php";
```

- lignes 4-6 : on définit l'objet **[$page]** ;
- ligne 8 : on inclut le script **[page-01.php]**. Le code de ce script va être interprété à son tour :
  - la variable **[$page]** est maintenant définie et l'interpréteur PHP va l'utiliser ;
  - le code HTML de **[page-01.php]** va être envoyé tel quel au client ;
  - les résultats des opérations PHP **[print]** vont être inclus dans le flux texte envoyé au client ;

Maintenant si nous exécutons le script web **[exemple-02.php]**, nous obtenons la chose suivante dans le navigateur :



Si nous visualisons le contenu texte reçu par le navigateur :



- les codes PHP qui étaient en **[2]** et **[3]** ont été remplacés par les résultats des deux commandes **[print]** ;

De cet exemple, on retiendra deux choses :

- les pages HTML destinées au navigateur peuvent être isolées dans des scripts PHP ne contenant que ce code HTML et quelques partie dynamiques générées par du code PHP. Il doit y avoir le moins de PHP possible dans ces pages ;
- toute la logique qui génère les données dynamiques incluses dans les pages HTML doit être isolée dans des scripts PHP purs, ne comportant aucun code de présentation des pages (HTML, CSS, Javascript…) ;

Cela autorise une séparation des tâches :

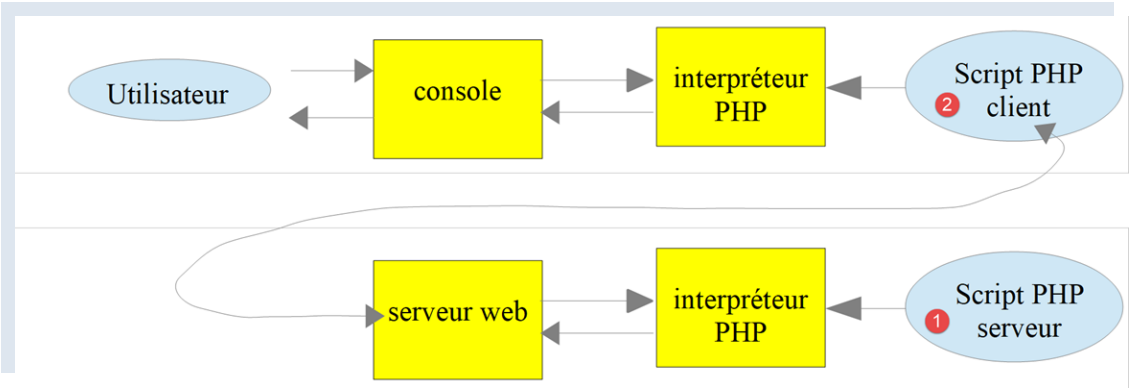
- la tâche de réalisation des pages web à afficher (HTML, CSS, Javascript…) ;
- la tâche de la logique de l'application web que l'on construit. Cette logique pourra être implémentée avec une architecture trois couches, exactement comme nous l'avons fait avec les scripts console ;

Par la suite, nous allons construire des scripts web particuliers ;

- ils n'enverront que des données au client et aucune décoration (HTML, CSS, Javascript). Ce seront donc des serveurs de données plutôt que des page web ;
- les clients de ces scripts web seront des scripts console qui se chargeront de récupérer les données envoyées par le serveur et d'en faire quelque chose ;

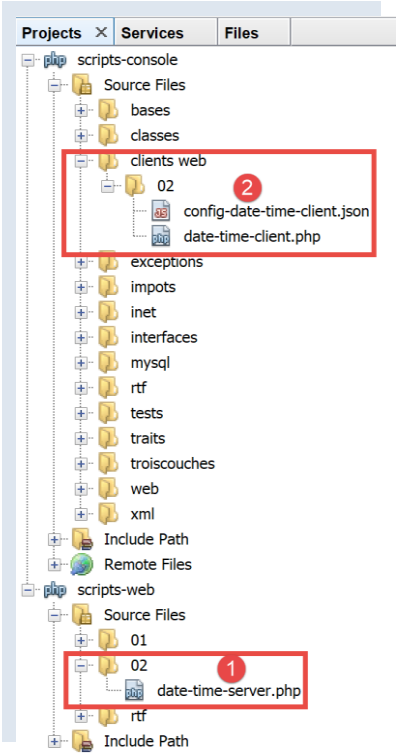
1.17.6 Application client/ serveur de date/heure

Nous nous plaçons maintenant dans la configuration suivante :



Nous allons écrire :

- un script web **[1]** qui envoie à son client la date et l'heure du moment présent ;
- un script console **[2]** qui sera le client du script web : il va récupérer la date et heure envoyées par le script web et les afficher sur la console ;



- en **[1]**, le script web **[date-time-server.php]** ;
- en **[2]**, le script console **[date-time-client]** client du script web ;

1.17.6.1 Le script serveur

Nous avons déjà écrit un script web générant la date et l'heure du moment présent au paragraphe lien. C'était le script **[exemple-01.php]** suivant :

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="UTF-8">
5.          <title>Exemple de page dynamique</title>
6.      </head>
7.      <body>
8.          <?php
9.          // time : nb de millisecondes depuis 01/01/1970
10.         // format affichage date-heure
11.         // d: jour sur 2 chiffres
12.         // m: mois sur 2 chiffres
13.         // y : année sur 2 chiffres
14.         // H : heure 0,23
15.         // i : minutes
16.         // s: secondes
17.         print "<b>Date et heure du jour : </b>" . date("d/m/y H:i:s", time());
18.         ?>
19.     </body>
20. </html>
```

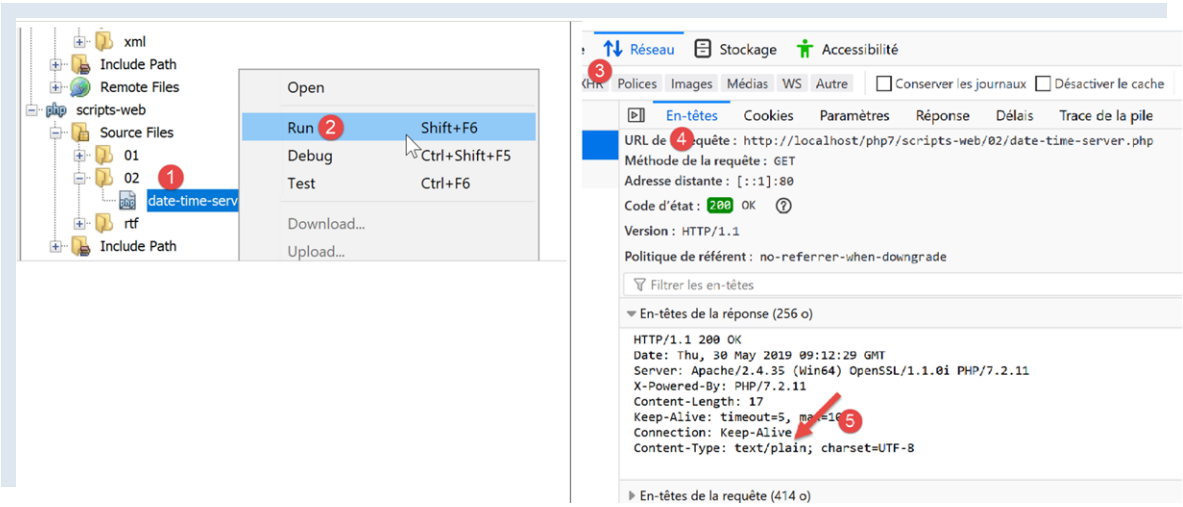
Nous avons dit que nous allions écrire des serveurs de données : des données brutes sans habillage HTML. Le script serveur **[date-time-server.php]** sera alors le suivant :

```
1.  <?php
2.
3.  // on fixe l'entête HTP [Content-Type]
4.  header('Content-Type: text/plain; charset=UTF-8');
5.  //
6.  // on envoie date et heure
7.  // time : nb de millisecondes depuis 01/01/1970
8.  // format affichage date-heure
9.  // d: jour sur 2 chiffres
10. // m: mois sur 2 chiffres
```

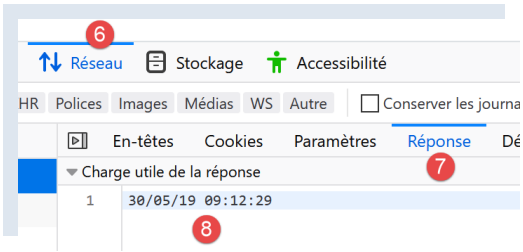
```
11. // y : année sur 2 chiffres
12. // H : heure 0,23
13. // i : minutes
14. // s: secondes
15. print date("d/m/y H:i:s", time());
```

- ligne 4 : on fixe l'entête HTTP **[Content-Type]** qui dit au client la nature du document qu'il va recevoir. Jusqu'à maintenant, le **[Content-Type]** était : **[Content-Type: text/html; charset=UTF-8]**. Ici, nous indiquons au client que le document est du texte sans habillage HTML. Ce n'est pas important pour notre client console qui ne cherchera pas à exploiter cet entête. C'est plus important pour les navigateurs clients qui eux exploitent cet entête ;

Exécutons ce script serveur :



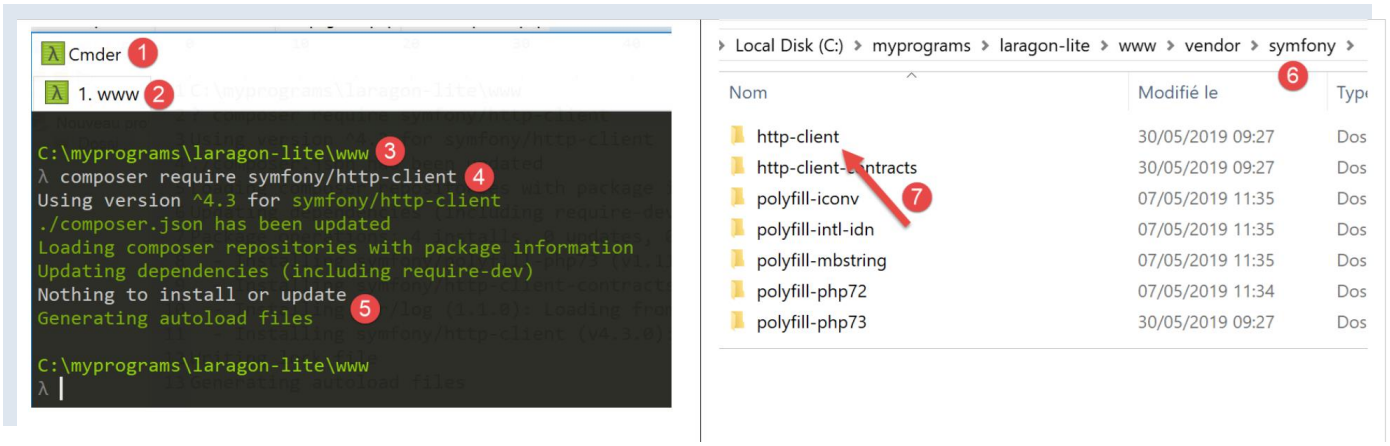
Si nous examinons dans le navigateur la réponse du serveur (F12), nous voyons en **[5]** l'entête HTTP que le script serveur a fixé et en **[8]**, le document texte reçu ;



1.17.6.2 Le script client

Au paragraphe lien nous avons développé plusieurs clients HTTP. Nous pourrions les utiliser pour récupérer le document texte envoyé par le script serveur **[date-time-server.php]**. Nous n'allons pas le faire. Comme nous l'avons fait pour les protocole SMTP et IMAP, nous allons utiliser une bibliothèque tierce, à savoir le composant **[HttpClient]** du framework Symfony **[https://symfony.com/doc/master/components/http_client.html]**.

Comme pour les deux précédentes bibliothèques, on utilise l'outil **[Composer]** pour installer le composant **[HttpClient]** de Symfony. Dans une fenêtre **[Terminal]** de Laragon (cf paragraphe lien), on tape la commande suivante :

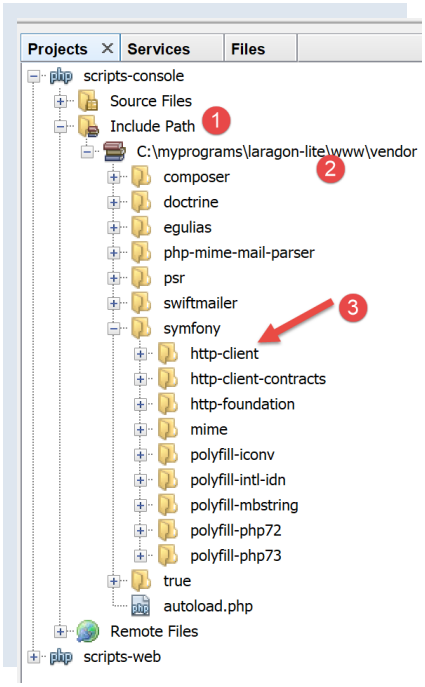


- en **[3]**, vérifiez que vous êtes positionné sur le dossier **[<laragon>/www/]** où <laragon> est le dossier d'installation de Laragon ;
- en **[4]**, la commande **[composer]** qui installe la bibliothèque **[HttpClient]** de Symfony ;
- en **[5]**, rien n'est installé car la bibliothèque **[HttpClient]** avait déjà été installée sur ce poste ;
- en **[6-7]**, de nouveaux dossiers apparaissent dans **[<laragon>/www/vendor/symfony]** ;

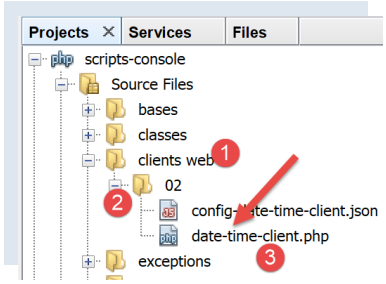
A la place de **[5]**, vous devriez avoir quelque chose comme suit :

```
1   C:\myprograms\laragon-lite\www
2   ? composer require symfony/http-client
3   Using version ^4.3 for symfony/http-client
4   ./composer.json has been updated
5   Loading composer repositories with package information
6   Updating dependencies (including require-dev)
7   Package operations: 4 installs, 0 updates, 0 removals
8     - Installing symfony/polyfill-php73 (v1.11.0): Downloading (100%)
9     - Installing symfony/http-client-contracts (v1.1.1): Downloading (100%)
10    - Installing psr/log (1.1.0): Loading from cache
11    - Installing symfony/http-client (v4.3.0): Downloading (100%)
12  Writing lock file
13  Generating autoload files
```

Assurez-vous que le dossier **[<laragon>/www/vendor]** fait partie de la branche **[Include Path]** de votre projet (cf paragraphe lien) :



Ceci fait, nous pouvons écrire le script console **[date-time-client.php]** :



Le script console **[date-time-client.php]** exploitera le fichier jSON **[config-date-time-client.json]** suivant :

```
1.  {
2.    "url": "http://localhost/php7/scripts-web/02/date-time-server.php"
3.  }
```

- ligne 2 : l'URL du script serveur ;

Le script client **[date-time-client.php]** sera le suivant :

```
1   <?php
2
3   // client du service date / heure
4   //
5   // gestion des erreurs
6   //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7   //ini_set("display_errors", "off");
8   //
9   // dépendances
10  require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11  use Symfony\Component\HttpClient\HttpClient;
12
13  // la configuration du client
14  const CONFIG_FILE_NAME = "config-date-time-client.json";
15
16  // on récupère la configuration
17  if (!file_exists(CONFIG_FILE_NAME)) {
18    print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
19    exit;
20  }
21  if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
22    print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
23    exit;
24  }
25
26  // on crée un client HTTP
27  $httpClient = HttpClient::create();
28
29  try {
30    // on fait la requête
31    $response = $httpClient->request('GET', $config['url']);
32    // statut de la réponse
33    $statusCode = $response->getStatusCode();
34    print "---Réponse avec statut : $statusCode\n";
35    // on récupère les entêtes
36    print "---Entêtes de la réponse\n";
37    $headers = $response->getHeaders();
38    foreach ($headers as $type => $value) {
39      print "$type: " . $value[0] . "\n";
40    }
41    // on récupère le corps de la réponse
42    $content = $response->getContent();
43    // on l'affiche
44    print "---Réponse du serveur : [$content]\n";
45  } catch (TypeError | RuntimeException $ex) {
46    // on affiche l'erreur
47    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
48    exit;
49  }
```

Commentaires

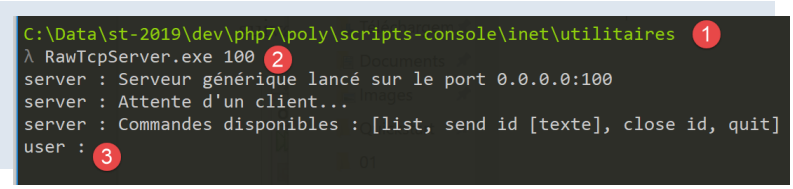
- ligne 10 : comme nous l'avions fait pour les bibliothèques précédentes, nous chargeons le fichier **[<laragon>/www/vendor/autoload.php]** ;
- ligne 11 : nous déclarons la classe **[HttpClient]** que nous allons utiliser ;
- lignes 13-24 : on récupère la configuration du script dans le dictionnaire **[$config]** ;
- ligne 27 : on crée un objet de type **[HttpClient]** ;
- ligne 31 : on demande l'URL du script serveur à l'aide d'une commande GET : **[GET URL HTTTP/1.1]**. Cette opération est asynchrone. L'exécution se poursuit en ligne 33 sans attendre que la réponse soit obenue ;
- ligne 33 : on demande le statut de la réponse. Ce statut se trouve dans le 1er entête HTTP renvoyé par le serveur. Ainsi si cet entête est **[HTTP/1.1 200 OK]**, le statut de la réponse est 200. Cette opération est bloquante : on n'en revient que lorsque le client a reçu toute la réponse du serveur ;
- ligne 37 : on demande les entêtes HTTP de la réponse ;
- ligne 42 : on demande le document renvoyé par le serveur : on sait que ce document est ici un texte.
- lignes 45-49 : en cas d'erreur, on affiche le message d'erreur ;

Lorsqu'on exécute le script client (il faut que Laragon soit lancé pour que le script serveur puisse être atteint), on obtient le résultat suivant sur la console :

```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Thu, 30 May 2019 14:42:03 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   content-length: 17
7   content-type: text/plain; charset=UTF-8
8   ---Réponse du serveur : [30/05/19 14:42:03]
```

On récupère bien la date et l'heure du moment présent en ligne 8.

On peut avoir la curiosité de savoir ce qu'à envoyé le script client au serveur. Pour cela nous allons utiliser notre serveur TCP générique (cf paragraphe lien) :



- en **[1]**, le dossier des utilitaires ;
- en **[2]**, le serveur TCP est lancé sur le port 100 ;
- en **[3]**, attente d'une commande tapée au clavier ;

Nous modifions le fichier de configuration du script **[date-time-client.php]** :

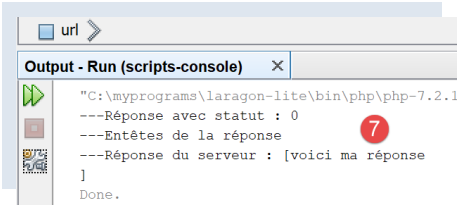
```
1   {
2       "url": "http://localhost:100/php7/scripts-web/02/date-time-server.php"
3   }
```

Cette fois-ci, le client contacte le serveur **[localhost]** sur le port 100. C'est donc notre serveur TCP générique qui va être sollicité. Lorsque nous exécutons le script console **[date-time-client.php]**, la console du serveur TCP générique évolue de la façon suivante :

```
C:\Data\st-2019\dev\php7\poly\scripts-console\inet\utilitaires
λ RawTcpServer.exe 100
server : Serveur générique lancé sur le port 0.0.0.0:100
server : Attente d'un client...
server : Commandes disponibles : [list, send id [texte], close id, quit]
user : server : Client 1-DESKTOP-528I5CU-54633 connecté...
server : Attente d'un client...
client 1 : [GET /php7/scripts-web/02/date-time-server.php HTTP/1.1] 3
client 1 : [Host: localhost:100]
client 1 : [User-Agent: Symfony HttpClient/Curl] 4
client 1 : [Accept-Encoding: deflate, gzip]
client 1 : []
send 1 [voici ma réponse] 5
user : close 1 6
server : Connexion client 1 fermée...
user :
```

- en **[3]**, la commande HTTP GET construite par le script client ;
- en **[4]**, la signature du script console ;
- en **[5]**, la réponse du serveur au script client. On notera que ce n'est pas une réponse HTTP valide :
  - il devrait y avoir des entêtes HTTP ;
  - puis une ligne vide ;
  - puis le document texte envoyé au client ;
- en **[6]**, on ferme la communication avec le script client pour que celui-ci détecte qu'il a eu la totalité de la réponse ;

Côté script client, on a l'affichage console suivant :

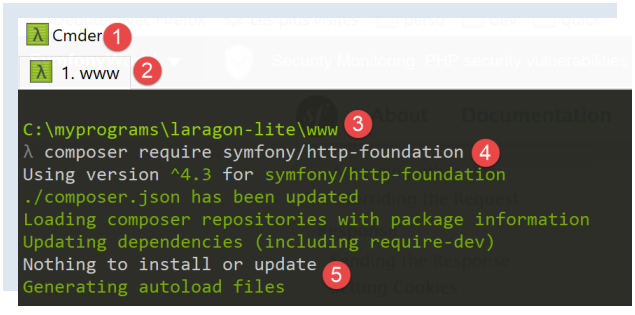


- en **[7]**, ce qu'a reçu le client Symfony ;

1.17.6.3 Le script serveur – version 2

De base, les fonctions PHP pour écrire un script web ne sont pas orientées objet. Côté serveur, on est alors amenés à mélanger classes et fonctions PHP classiques. Pour avoir une écriture plus homogène, nous allons utiliser la bibliothèque **[HttpFoundation]** du framework Symfony. Elle a encapsulé toutes les fonctions PHP classiques pour un service web dans un système de classes et interfaces. La documentation de la bibliothèque est disponible à l'URL **[https://symfony.com/doc/current/components/http_foundation.html]** (mai 2019).

Pour installer la bibliothèque, nous procédons de la façon suivante dans un terminal Laragon (cf paragraphe lien) :



- **[2-3]** : assurez-vous d'être positionné dans le dossier **[<laragon>/www]** ;
- **[4]** : la commande **[composer]** qui va installer la bibliothèque **[HttpFoundation]** ;
- **[5]** : sur cet exemple, la bibliothèque était déjà installée ;

A la première installation, vous devriez avoir des logs console ressemblant à ceci :

```
C:\myprograms\laragon-lite\www
? composer require symfony/http-foundation
Using version ^4.3 for symfony/http-foundation
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 2 installs, 0 updates, 0 removals
  - Installing symfony/mime (v4.3.0): Downloading (100%)
  - Installing symfony/http-foundation (v4.3.0): Downloading (100%)
Writing lock file
Generating autoload files
```

La seconde version du serveur web **[date-time-server-2.php]** est la suivante :

```
<?php

// usage des bibliothèques de Symfony

// dépendances
require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
use Symfony\Component\HttpFoundation\Response;

// on fixe l'entête Content-Type
$response=new Response();
$response->headers->set("content-type","text/plain");
$response->setCharset("utf-8");

// on fixe le contenu de la réponse
//
// on envoie date et heure
// time : nb de millisecondes depuis 01/01/1970
// format affichage date-heure
// d: jour sur 2 chiffres
// m: mois sur 2 chiffres
// y : année sur 2 chiffres
// H : heure 0,23
// i : minutes
// s: secondes
$response->setContent(date("d/m/y H:i:s", time()));

// on envoie la réponse
$response->send();
```

Commentaires

- ligne 7 : la classe **[Response]** de la bibliothèque **[HttpFoundation]** de Symfony gère la totalité de la réponse aux clients du service web ;
- ligne 10 : création d'une instance de la classe **[Response]** ;
- ligne 11 : on indique que la réponse est de type **[text/plain]** ;
- ligne 12 : la réponse est du texte UTF-8 ;
- ligne 25 : on fixe le document de la réponse, ce qu'a demandé le client ;
- ligne 28 : on envoie la réponse au client ;

1.17.6.4 Le script client – version 2

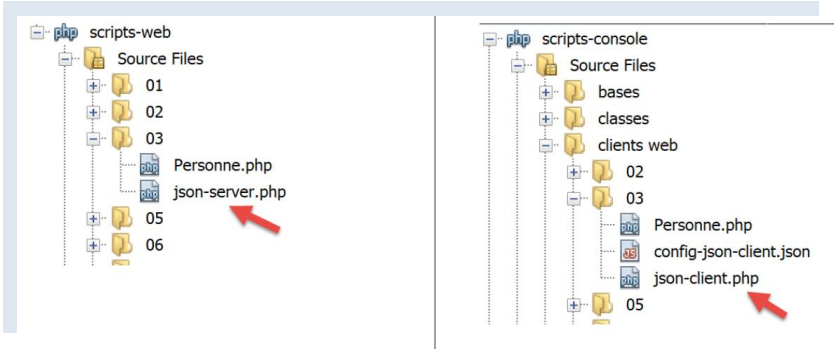
Le script client ne change pas. On change seulement son fichier de configuration **[config-date-time-client.json]** :

```
{
    "url": "http://localhost/php7/scripts-web/02/date-time-server-2.php"
}
```

Les résultats sont les mêmes que dans la version 1.

1.17.7 Un serveur de données jSON

La réponse d'un script web peut être composée de plusieurs données qu'on peut rassembler dans des tableaux et des objets. Le script peut alors envoyer ces divers éléments au sein d'une chaîne jSON que le client décodera.



1.17.7.1 Le script serveur

Le script **[json-server.php]** utilise la classe **[Personne]** suivante :

```
1   <?php
2
3   namespace Modèles;
4
5   class Personne implements \JsonSerializable {
6     // attributs
7     private $nom;
8     private $prénom;
9     private $âge;
10
11    // conversion d'un tableau associatif vers un objet [Personne]
12    public function setFromArray(array $assoc): Personne {
13      // on initialise l'objet courant avec le tableau associatif
14      foreach ($assoc as $attribute => $value) {
15        $this->$attribute = $value;
16      }
17      // résultat
18      return $this;
19    }
20
21    // getters et setters
22    public function getNom() {
23      return $this->nom;
24    }
25
26    public function getPrénom() {
27      return $this->prénom;
28    }
29
30    public function setNom($nom) {
31      $this->nom = $nom;
32      return $this;
33    }
34
35    public function setPrénom($prénom) {
36      $this->prénom = $prénom;
37      return $this;
38    }
39
40    public function getÂge() {
41      return $this->âge;
42    }
43
44    public function setÂge($âge) {
45      $this->âge = $âge;
46      return $this;
47    }
48
49    // toString
50    public function __toString(): string {
51      return "Personne [$this->prénom, $this->nom, $this->âge]";
52    }
53
54    // implémente l'interface JsonSerializable
55    public function jsonSerialize(): array {
```

```
56      // on rend un tableau associatif avec pour clés les attributs de l'objet
57      // ce tableau pourra ensuite être encodé en jSON
58      return get_object_vars($this);
59    }
60
61    // conversion d'un jSON vers un objet [Personne]
62    public static function jsonUnserialize(string $json): Personne {
63      // on crée une personne à partir de la chaîne jSON
64      return (new Personne())->setFromArray(json_decode($json, true));
65    }
66
67  }
```

Commentaires

- ligne 5 : la classe implémente l'interface PHP **[JsonSerializable]**. Cela lui impose d'implémenter la méthode **[jsonSerialize]** des lignes 55-59. La méthode doit rendre un tableau associatif qui devra être sérialisé en jSON. Lorsqu'on utilise l'expression **[json_encode($personne)]**, la fonction **[json_encode]** regarde si la classe **[Personne]** implémente l'interface **[JsonSerializable]**. Si oui, l'expression devient **[json_encode($personne→serialize())]** ;
- lignes 12-19 : la classe n'a pas de constructeur mais un initialiseur. La classe **[Personne]** peut être alors instanciée par l'expression **[(new Personne())→setFromArray($array)]**. On peut avoir divers types d'initialiseurs alors qu'on ne peut avoir qu'un constructeur. Ces initialiseurs permettent divers modes d'instanciation du type **[(new Personne())→initialiseuri(…)]** ;
- lignes 62-65 : la fonction statique **[jsonUnserialize]** permet de créer un objet **[Personne]** à partir de sa chaîne jSON ;

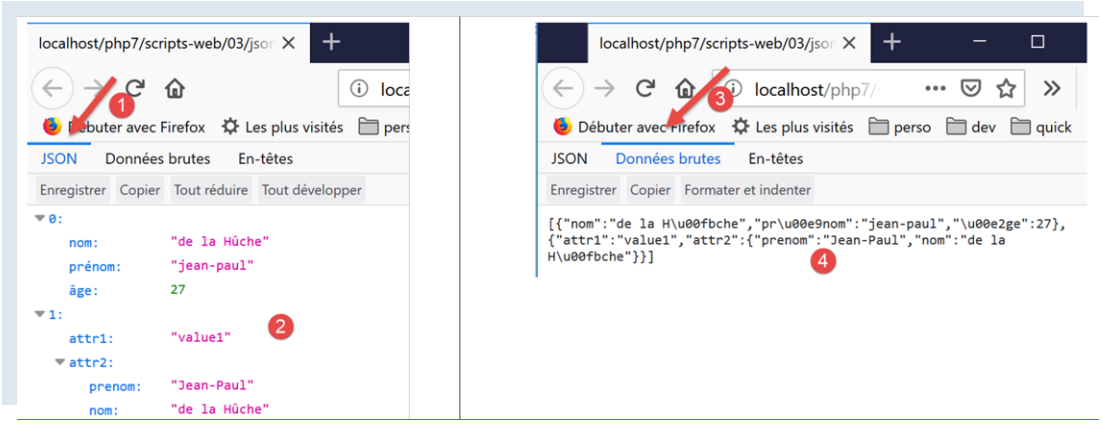
Le script **[json-server.php]** sera le suivant :

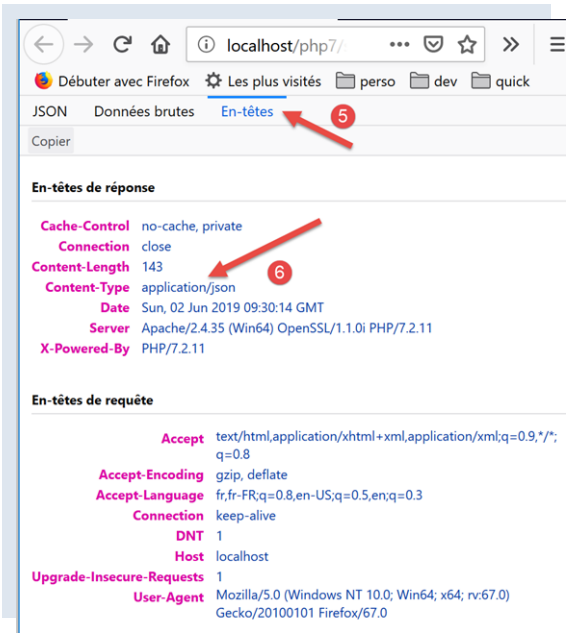
```
1   <?php
2
3   // dépendances
4   require_once __DIR__ . "/Personne.php";
5   use \Modèles\Personne;
6   require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
7   use \Symfony\Component\HttpFoundation\Response;
8
9   // on fixe l'entête Content-Type et la bibliothèque de caractères utilisée
10  $response = new Response();
11  $response->headers->set("content-type", "application/json");
12  $response->setCharset("utf-8");
13
14  // on crée un objet Personne
15  $personne = (new Personne())->setFromArray([
16    "nom" => "de la Hûche",
17    "prénom" => "jean-paul",
18    "âge" => 27]);
19  // un tableau associatif
20  $assoc = ["attr1" => "value1",
21    "attr2" => [
22      "prenom" => "Jean-Paul",
23      "nom" => "de la Hûche"
24    ]
25  ];
26  // le contenu de la réponse est du jSON
27  $response->setContent(json_encode([$personne, $assoc]));
28
29  // envoi de la réponse
30  $response->send();
```

Commentaires

- lignes 4-5 : on importe la classe **[Personne]** ;
- ligne 11 : on indique que le document sera de type **[application/json]**. A la réception de cet entête, les navigateurs afficheront une mise en forme de la chaîne jSON plutôt que d'afficher du texte brut ;
- ligne 12 : la chaîne jSON contiendra des caractères UTF-8 ;
- lignes 15-18 : on crée un objet **[Personne]** ;
- lignes 20-25 : on crée un tableau associatif à deux niveaux ;
- ligne 27 : on envoie au client la chaîne jSON d'un tableau :
  - l'élément **[$personne]** sera sérialisée en jSON grâce à sa méthode **[jsonSerialize]** ;
  - l'élément **[$assoc]** sera nativement sérialisé en jSON ;

Si on exécute ce script serveur (Laragon doit être lancé), on obtient la réponse suivante dans un navigateur :

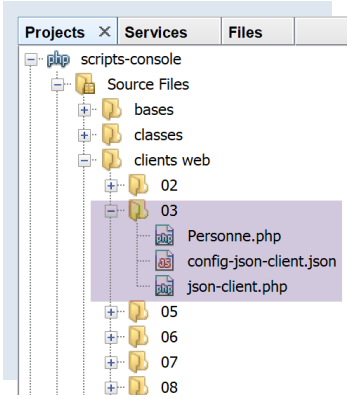




Commentaires

- en **[2]**, la réponse jSON mise en forme ;
- en **[4]**, la réponse jSON brute. On remarquera l'encodage des caractères accentués ;
- en **[6]**, c'est le type de contenu **[application/json]** envoyé par le serveur qui a conduit le navigateur à faire cette mise en forme ;

1.17.7.2 Le client



Le client **[json-client.php]** est configuré par le fichier jSON **[config-json-client.json]** suivant :

```
{
    "url": "http://localhost/php7/scripts-web/03/json-server.php"
}
```

Le script **[json-client.php]** est le suivant :

```
<?php

// client d'un service jSON
//
// gestion des erreurs
//ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
//ini_set("display_errors", "off");
//
// dépendances
require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
use Symfony\Component\HttpClient\HttpClient;
require_once __DIR__ . "/Personne.php";
use \Modèles\Personne;

// la configuration du client
const CONFIG_FILE_NAME = "config-json-client.json";

// on récupère la configuration
if (!file_exists(CONFIG_FILE_NAME)) {
  print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
  exit;
}
if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
  print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
  exit;
}

// on crée un client HTTP
$httpClient = HttpClient::create();

try {
  // on fait la requête
  $response = $httpClient->request('GET', $config['url']);
  // statut de la réponse
  $statusCode = $response->getStatusCode();
  print "---Réponse avec statut : $statusCode\n";
  // on récupère les entêtes
  print "---Entêtes de la réponse\n";
  $headers = $response->getHeaders();
  foreach ($headers as $type => $value) {
    print "$type: " . $value[0] . "\n";
  }
  // on récupère le corps jSON de la réponse
  list($personne, $assoc) = json_decode($response->getContent(), true);
  // on instancie une personne à partir du tableau de ses attributs
  $personne = (new Personne())->setFromArray($personne);
  // on affiche la réponse du serveur
  print "---Réponse du serveur\n";
  print "$personne\n";
  print "tableau=" . json_encode($assoc, JSON_UNESCAPED_UNICODE) . "\n";
} catch (TypeError | RuntimeException $ex) {
```

```
52    // on affiche l'erreur
53    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
54  }
55
```

Commentaires

- lignes 12-13 : importation de la classe **[Personne]** ;
- ligne 30 : création du client HTTP ;
- ligne 44 : on décode la chaîne jSON envoyée par le serveur. On sait que ce qui a été encodé est un tableau à deux éléments comprenant deux tableaux associatifs ;
- ligne 46 : on crée un objet **[Personne]** pour l'afficher ensuite en ligne 49 ;
- ligne 50 : on affiche le 2e tableau associatif. L'instruction **[print]** ne sait pas afficher des tableaux. Aussi transforme-t-on celui-ci en chaîne jSON. Pour avoir correctement les caractères accentués, il faut mettre le second paramètre **[JSON_UNESCAPED_UNICODE]**. On a vu qu'effectivement les caractères accentués sont encodés dans la chaîne jSON ;

L'exécution du script client donne les résultats suivants :

```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Sun, 02 Jun 2019 09:56:29 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: no-cache, private
7   content-length: 143
8   connection: close
9   content-type: application/json
10  ---Réponse du serveur
11  Personne [jean-paul, de la Hûche, 27]
12  tableau={"attr1":"value1","attr2":{"prenom":"Jean-Paul","nom":"de la Hûche"}}
```

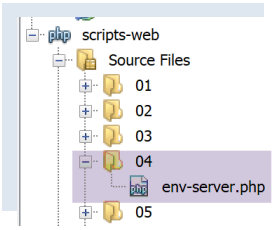
Lignes 11 et 12, on a récupéré correctement les caractères accentués.

1.17.8 Récupération des variables d'environnement du service web

Un script serveur s'exécute dans un environnement web qu'il peut connaître. Cet environnement est stocké dans le dictionnaire $_SERVER, une variable globale de PHP. Si nous utilisons la bibliothèque **[HttpFoundation]**, cet environnement sera trouvé dans le champ **[Request→server]** où **[Request]** est la requête HTTP traitée par le script web.

1.17.8.1 Le script serveur

Nous écrivons une application serveur qui envoie à ses clients son environnement d'exécution.



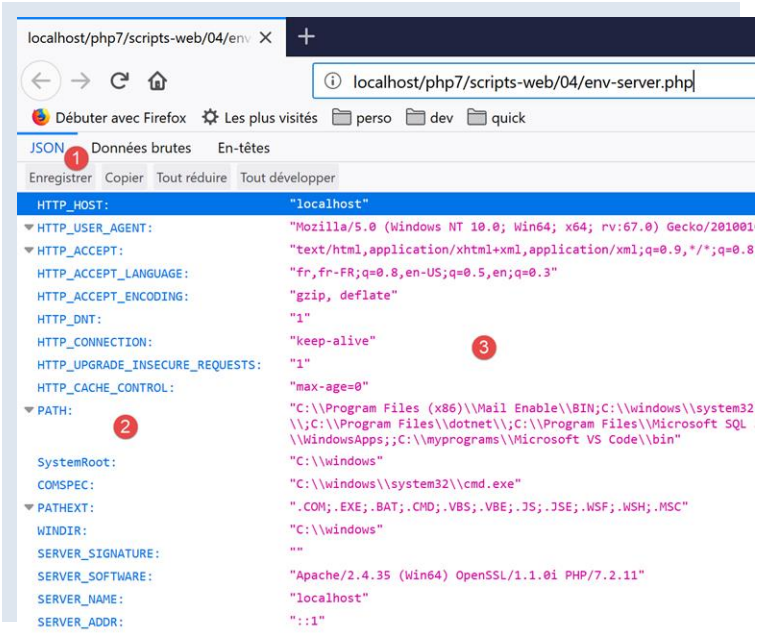
Le script web **[env-server.php]** est le suivant :

```
1   <?php
2
3   // dépendances
4   require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
5   use \Symfony\Component\HttpFoundation\Response;
6   use Symfony\Component\HttpFoundation\Request;
7
8   // on récupère la requête
9   $request = Request::createFromGlobals();
10  // on élabore la réponse
11  $response = new Response();
12  // le contenu de la réponse est du json utf-8
13  $response->headers->set("content-type", "application/json");
14  $response->setCharset("utf-8");
15  // on fixe le contenu jSON de la réponse
```

```
16  $response->setContent(json_encode($request->server->all()));
17  // envoi de la réponse
18  $response->send();
```

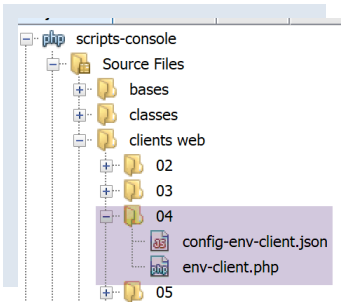
- ligne 9 : on récupère l'objet de type **[Request]** qui encapsule la totalité des informations disponibles sur la requête HTTP reçue par le script web ainsi que sur l'environnement d'exécution de celui-ci ;
- lignes 13-14 : on va envoyer du texte brut avec caractères UTF-8 au client ;
- ligne 16 : l'information envoyée au client sera une chaîne de caractères obtenue par sérialisation jSON de l'objet **[$request→server→all()]** : **[$request→server]** représente l'environnement d'exécution du script web. C'est un objet de type **[ServerBag]**, une sorte de dictionnaire. **[$request→server→all()]** est lui un vrai dictionnaire, celui du contenu du **[ServerBag]** ;
- ligne 18 : on envoie l'information ;

Si on exécute ce script à partir de Netbeans, le navigateur affiche la page suivante :



- en **[2]**, les différentes clés du dictionnaire de l'environnement ;
- en **[3]**, les valeurs de ces clés ;

1.17.8.2 Le script client



Le script client **[env-client.php]** est configuré par le fichier jSON **[config-env-client.json]** suivant :

```
1   {
2       "url": "http://localhost/php7/scripts-web/04/env-server.php"
3   }
```

Le script client **[env-client.php]** est le suivant :

```
1   <?php
2
3   // environnement d'un script serveur
```

```
4   //
5   // gestion des erreurs
6   //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7   //ini_set("display_errors", "off");
8   //
9   // dépendances
10  require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11  use Symfony\Component\HttpClient\HttpClient;
12
13  // la configuration du client
14  const CONFIG_FILE_NAME = "config-env-client.json";
15
16  // on récupère la configuration
17  if (!file_exists(CONFIG_FILE_NAME)) {
18    print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
19    exit;
20  }
21  if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
22    print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
23    exit;
24  }
25
26  // on crée un client HTTP
27  $httpClient = HttpClient::create();
28  try {
29    // on fait la requête au serveur
30    $response = $httpClient->request('GET', $config['url']);
31    // statut de la réponse
32    $statusCode = $response->getStatusCode();
33    print "---Réponse avec statut : $statusCode\n";
34    // on récupère les entêtes
35    print "---Entêtes de la réponse\n";
36    $headers = $response->getHeaders();
37    foreach ($headers as $type => $value) {
38      print "$type: " . $value[0] . "\n";
39    }
40    // on affiche la réponse du serveur
41    print "---Réponse du serveur\n";
42    $env = json_decode($response->getContent());
43    foreach ($env as $key => $value) {
44      print "[$key]=>$value\n";
45    }
46  } catch (TypeError | RuntimeException $ex) {
47    // on affiche l'erreur
48    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
49  }
```

Commentaires

- ligne 42 : on désérialise la réponse jSON du serveur. On obtient un tableau associatif ;
- lignes 43-45 : on affiche toutes les valeurs de ce tableau associatif ;

On obtient le résultat console suivant :

```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Sun, 02 Jun 2019 17:35:50 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: no-cache, private
7   content-length: 1505
8   connection: close
9   content-type: application/json
10  ---Réponse du serveur
11  [HTTP_HOST]=>localhost
12  [HTTP_USER_AGENT]=>Symfony HttpClient/Curl
13  [HTTP_ACCEPT_ENCODING]=>deflate, gzip
14  [PATH]=>C:\Program Files (x86)\Mail
    Enable\BIN;C:\windows\system32;C:\windows;C:\windows\System32\Wbem;C:\windows\System32\WindowsPowerShell\v1
    .0\;C:\windows\System32\OpenSSH\;C:\Program Files\dotnet\;C:\Program Files\Microsoft SQL
    Server\130\Tools\Binn\;C:\Program Files (x86)\Mail
    Enable\BIN64;C:\Users\serge\AppData\Local\Microsoft\WindowsApps;;C:\myprograms\Microsoft VS Code\bin
15  [SystemRoot]=>C:\windows
16  [COMSPEC]=>C:\windows\system32\cmd.exe
17  [PATHEXT]=>.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
18  [WINDIR]=>C:\windows
```

```
[SERVER_SIGNATURE]=>
[SERVER_SOFTWARE]=>Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
[SERVER_NAME]=>localhost
[SERVER_ADDR]=>::1
[SERVER_PORT]=>80
[REMOTE_ADDR]=>::1
[DOCUMENT_ROOT]=>C:/myprograms/laragon-lite/www
[REQUEST_SCHEME]=>http
[CONTEXT_PREFIX]=>
[CONTEXT_DOCUMENT_ROOT]=>C:/myprograms/laragon-lite/www
[SERVER_ADMIN]=>admin@example.com
[SCRIPT_FILENAME]=>C:/myprograms/laragon-lite/www/php7/scripts-web/04/env-server.php
[REMOTE_PORT]=>63744
[GATEWAY_INTERFACE]=>CGI/1.1
[SERVER_PROTOCOL]=>HTTP/1.1
[REQUEST_METHOD]=>GET
[QUERY_STRING]=>
[REQUEST_URI]=>/php7/scripts-web/04/env-server.php
[SCRIPT_NAME]=>/php7/scripts-web/04/env-server.php
[PHP_SELF]=>/php7/scripts-web/04/env-server.php
[REQUEST_TIME_FLOAT]=>1559496950.644
[REQUEST_TIME]=>1559496950
```

Voici la signification de certaines des variables (pour windows. Sous Linux, elles seraient différentes) :

`HTTP_HOST`	la valeur xxx de l'entête HTTP **`[Host: xxx]`** envoyée par le client
`HTTP_USER_AGENT`	la valeur xxx de l'entête HTTP **`[User_Agent: xxx]`** envoyée par le client
`HTTP_ACCEPT_ENCODING`	la valeur xxx de l'entête HTTP **`[Accept-Encoding: xxx]`** envoyée par le client
`PATH`	le chemin des exécutables sur la machine sur laquelle s'exécute le script serveur
`COMSPEC`	le chemin de l'interpréteur de commandes DOS
`PATHEXT`	les extensions des fichiers exécutables
`WINDIR`	le dossier d'installation de Windows
`SERVER_SIGNATURE`	la signature du serveur web. Ici rien.
`SERVER_SOFTWARE`	le type du serveur web
`SERVER_NAME`	le nom Internet de la machine du serveur web
`SERVER_PORT`	le port d'écoute du serveur web
`SERVER_ADDR`	l'adresse IP de la machine du serveur web, ici 127:0:0:1
`REMOTE_ADDR`	l'adresse IP du client. Ici le client était sur la même machine que le serveur.
`REMOTE_PORT`	le port de communication du client
`DOCUMENT_ROOT`	la racine de l'arborescence des documents servis par le serveur web
`REQUEST_SCHEME`	le protocole TCP de la requête d'URL http://localhost/php7/…
`SERVER_ADMIN`	l'adresse électronique de l'administrateur du serveur web
`SCRIPT_FILENAME`	le chemin complet du script serveur
`REMOTE_PORT`	le port à partir duquel le client a fait sa demende
`SERVER_PROTOCOL`	la version du protocole HTTP utilisée par le serveur web
`REQUEST_METHOD`	l'ordre HTTP utilisé par le client. Il y en a quatre : GET, POST, PUT, DELETE
`QUERY_STRING`	les paramètres envoyés avec un ordre GET /url?paramètres
`REQUEST_URI`	l'URL demandée par le client. Si le navigateur demande l'URL http://machine**`[:port]`**/uri, on aura REQUEST_URI=uri
`SCRIPT_NAME`	$_SERVER**`['SCRIPT_FILENAME']`**=$_SERVER**`['DOCUMENT_ROOT']`**.$_SERVER**`['SCRIPT_NAME']`**

1.17.9 Récupération par le serveur de paramètres envoyés par un client

1.17.9.1 Introduction

Dans le protocole HTTP, un client a deux méthodes pour passer des paramètres au serveur WEB :

- il demande l'URL du service sous la forme
  GET *url?param1=val1&param2=val2&param3=val3… HTTP/1.0*

où les valeurs *vali* doivent au préalable subir un encodage afin que certains caractères réservés soient remplacés par leur valeur hexadécimale ;

- il demande l'URL du service sous la forme
  POST *url HTTP/1.0*

puis parmi les entêtes HTTP envoyés au serveur place l'entête suivant :

Content-length=N

La suite des entêtes envoyés par le client se terminent par une ligne vide. Il peut alors envoyer ses données sous la forme

val1&param2=val2&param3=val3…

où les valeurs *vali* doivent, comme pour la méthode GET, être préalablement encodées. Le nombre de caractères envoyés au serveur doit être N où N est la valeur déclarée dans l'entête

Content-length=N

Le script PHP du service web qui récupère les paramètres *parami* précédents envoyés par le client obtient leurs valeurs dans le tableau :

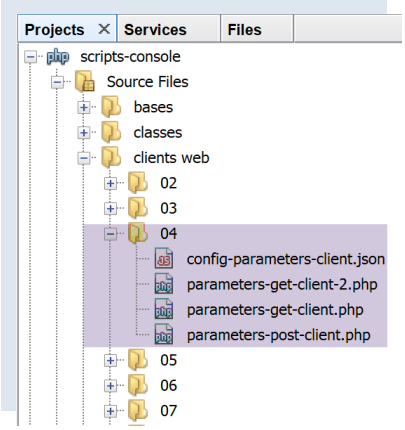
- $_GET**["parami"]** pour une commande GET ;
- $_POST**["parami"]** pour une commande POST ;

ceci pour les fonctions de base de PHP. Si on utilise la bibliothèque **[HttpFoundation]**, ces paramètres seront trouvés dans :

- **[Request]->query->get**('parami') pour une commande GET ;
- **[Request]->request->get**('parami') pour une commande POST ;

où **[Request]** représente la totalité des informations sur la requête reçue par le script web ;

1.17.9.2 Le client GET – version 1



Les scripts clients sont configurés par le fichier jSON **[config-parameters-client.json]** suivant :

```
1   {
2       "url-get": "http://localhost/php7/scripts-web/05/parameters-server.php",
3       "url-post": "http://localhost/php7/scripts-web/05/parameters-server.php"
4   }
```

- ligne 1 : l'URL du script web cible des clients GET ;
- ligne 2 : l'URL du script web cible du client POST ;

Les clients GET envoient trois paramètres **[nom, prenom, age]** au serveur. Le client **[parameters-get-client.php]** est le suivant :

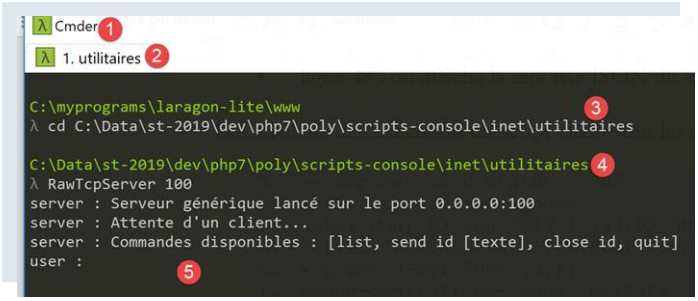
```
1   <?php
2
3   // client GET d'un serveur web
4   //
5   // gestion des erreurs
6   //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7   //ini_set("display_errors", "off");
8   //
```

```
9   // dépendances
10  require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11  use Symfony\Component\HttpClient\HttpClient;
12
13  // la configuration du client
14  const CONFIG_FILE_NAME = "config-parameters-client.json";
15
16  // on récupère la configuration
17  if (!file_exists(CONFIG_FILE_NAME)) {
18    print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
19    exit;
20  }
21  if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
22    print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
23    exit;
24  }
25
26  // on crée un client HTTP
27  $httpClient = HttpClient::create();
28
29  try {
30    // on prépare les paramètres
31    list($prenom, $nom, $age) = array("jean-paul", "de la hûche", 45);
32  // on encode les informations
33    $parameters = "prenom=" . urlencode($prenom) .
34      "&nom=" . urlencode($nom) .
35      "&age=$age";
36    // on fait la requête
37    $response = $httpClient->request('GET', $config['url-get'] . "?$parameters");
38    // statut de la réponse
39    $statusCode = $response->getStatusCode();
40    print "---Réponse avec statut : $statusCode\n";
41    // on récupère les entêtes
42    print "---Entêtes de la réponse\n";
43    $headers = $response->getHeaders();
44    foreach ($headers as $type => $value) {
45      print "$type: " . $value[0] . "\n";
46    }
47    // on affiche la réponse du serveur
48    print "---Réponse du serveur [" . $response->getContent() . "]\n";
49  } catch (TypeError | RuntimeException $ex) {
50    // on affiche l'erreur
51    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
52  }
```

Commentaires

- lignes 33-35 : encodage des paramètres envoyés au serveur. Les paramètres **[$prenom, $nom]** qui peuvent avoir des caractères UTF-8 sont encodés avec la fonction **[urlencode]**. Tous les caractères non alphanumériques (au sens des expressions relationnelles) sont remplacés par %xx où xx est la valeur hexadécimale du caractère. Les espaces sont eux remplacés par le signe + ;
- ligne 37 : l'URL demandée est **$URL?$parameters** où $*parameters* est de la forme *nom=val1&prenom=val2&age=val3* ;
- ligne 48 : le client se contentera d'afficher a réponse du client ;

On peut avoir la curiosité de voir ce que reçoit le serveur lors d'une requête GET paramétrée. Pour cela, nous lançons notre serveur générique **[RawTcpServer]** sur le port 100 de la machine locale à partir d'un terminal Laragon (cf paragraphe lien) :



Vérifiez qu'en **[4]**, vous êtes bien dans le dossier des utilitaires.

Nous modifions le fichier jSON **[parameters-get-client.json]** qui configure les clients GET et POST :

```
{
    "url-get": "http://localhost:100/php7/scripts-web/05/parameters-server.php",
    "url-post": "http://localhost/php7/scripts-web/05/parameters-server.php"
}
```

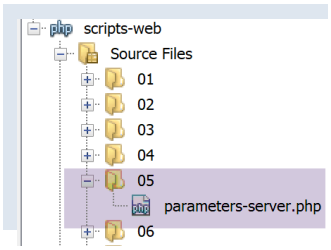
- ligne 2 : nous avons changé le port du serveur web. Ce sera donc **[RawTcpServer]** qui sera contacté ;

Nous exécutons le client. Dans la fenêtre de **[RawTcpServer]** nous obtenons les informations suivantes :

```
C:\Data\st-2019\dev\php7\poly\scripts-console\inet\utilitaires
λ RawTcpServer 100
server : Serveur générique lancé sur le port 0.0.0.0:100
server : Attente d'un client...
server : Commandes disponibles : [list, send id [texte], close id, quit]
user : server : Client 1-DESKTOP-528I5CU-52370 connecté...
server : Attente d'un client...
client 1 : [GET /php7/scripts-web/05/parameters-server.php?prenom=jean-paul&nom=de%20la%20h%C3%BBche&age=45 HTTP/1.1]
client 1 : [Host: localhost:100]
client 1 : [User-Agent: Symfony HttpClient/Curl]
client 1 : [Accept-Encoding: deflate, gzip]
client 1 : []
quit
server : fin du service
```

- en **[1]**, la commande GET paramétrée envoyée par le client. On voit clairement l'encodage de certains caractères ;

1.17.9.3 Le serveur GET / POST



Le script serveur **[parameters-server.php]** est le suivant :

```
<?php

// dépendances
require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
use \Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;

// on récupère la requête
$request = Request::createFromGlobals();
// on récupère les paramètres de la requête
$getParameters = $request->query->all();
$bodyParameters = $request->request->all();

// on élabore la réponse
$response = new Response();
// le contenu de la réponse est du texte utf-8
$response->headers->set("content-type", "application/json");
$response->setCharset("utf-8");
// contenu de la réponse - un tableau encodé en jSON
$response->setContent(json_encode([
  "method" => $request->getMethod(),
  "uri" => $request->getRequestUri(),
  "getParameters" => $getParameters,
  "bodyParameters" => $bodyParameters
    ], JSON_UNESCAPED_UNICODE));
// envoi de la réponse
$response->send();
```

Commentaires

- ligne 9 : création de l'objet **[Request]** du script web. Cet objet encapsule la totalité des informations que le script web a reçues du client ;
- ligne 11 : l'objet **[Request→query]** est de type **[ParameterBag]** et rassemble les paramètres de l'éventuelle opération GET d'un client. L'expression **[Request→query→get(«X»)]** permet d'avoir le paramètre nommé X dans les paramètres du GET **[nom=val1&prenom=val2&age=val3]**. L'expression **[Request→query→all()]** permet d'avoir le dictionnaire des paramètres du GET ;
- ligne 12 : l'objet **[Request→request]** est de type **[ParameterBag]** et rassemble les paramètres envoyés comme document du client au serveur. On dit également de ces paramètres qu'ils sont uploadés parce qu'ils appartiennent à un document que le client envoie au serveur. L'expression **[Request→request→get(«X»)]** permet d'avoir le paramètre nommé X dans les paramètres uploadés **[nom=val1&prenom=val2&age=val3]**. L'expression **[Request→request→all()]** permet d'avoir le dictionnaire des paramètres uploadés ;
- lignes 17-18 : on indique au client qu'on va lui envoyer du jSON codé en UTF-8 ;
- lignes 20-25 : le serveur renvoie au client tous les paramètres qu'il a reçus ainsi, le type d'opération **[GET / POST / …]** faite par le client, et l'URI demandée. Cette méthode est obtenue par l'expression **[$request→getMethod()]**. Le document envoyé au client est la chaîne jSON d'un tableau associatif dont certaines valeurs sont eux-mêmes des tableaux associatifs. Le paramètre **[JSON_UNESCAPED_UNICODE]** demande à ce que les caractères Unicode (comme les caractères accentués par exemple) soient envoyés tels quels et pas encodés ;
- ligne 27 : la réponse est envoyée au client ;

L'exécution du script client donne les résultats suivants :

```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Mon, 03 Jun 2019 10:08:45 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: no-cache, private
7   content-length: 207
8   connection: close
9   content-type: application/json
10  ---Réponse du serveur [{"method":"GET","uri":"\/php7\/scripts-web\/05\/parameters-server.php?prenom=jean-
    paul&nom=de+la+h%C3%BBche&age=45","getParameters":{"prenom":"jean-paul","nom":"de la
    hûche","age":"45"},"bodyParameters":[]}]
```

- ligne 10 :
  - **[method]** : la méthode est GET ;
  - **[uri]** : on voit les paramètres url-encodés de la requête GET dans l'URI demandée ;
  - **[getParameters]** : le tableau des paramètres du GET ;
  - **[bodyParameters]** : le tableau des paramètres uploadés : il est vide ;

1.17.9.4 Le client GET – version 2

Dans la version précédente du script client, nous avons url-encodé nous-mêmes les paramètres envoyés au serveur, dans un but pédagogique. L'objet **[HttpClient]** sait faire ce travail lui-même. C'est le script **[parameters-get-client-2.php]** suivant :

```
1   <?php
2
3   // client GET d'un serveur web
4   //
5   // gestion des erreurs
6   //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7   //ini_set("display_errors", "off");
8   //
9   // dépendances
10  require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11  use Symfony\Component\HttpClient\HttpClient;
12
13  // la configuration du client
14  const CONFIG_FILE_NAME = "config-parameters-client.json";
15
16  // on récupère la configuration
17  if (!file_exists(CONFIG_FILE_NAME)) {
18    print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
19    exit;
20  }
21  if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
22    print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
23    exit;
24  }
25
26  // on crée un client HTTP
```

```
27  $httpClient = HttpClient::create();
28  try {
29    // on prépare les paramètres
30    list($prenom, $nom, $age) = array("jean-paul", "de la hûche", 45);
31    // on fait la requête au serveur
32    $response = $httpClient->request('GET', $config['url-get'],
33      ["query" => [
34          "prenom" => $prenom,
35          "nom" => $nom,
36          "age" => $age
37    ]]);
38    // statut de la réponse
39    $statusCode = $response->getStatusCode();
40    print "---Réponse avec statut : $statusCode\n";
41    // on récupère les entêtes
42    print "---Entêtes de la réponse\n";
43    $headers = $response->getHeaders();
44    foreach ($headers as $type => $value) {
45      print "$type: " . $value[0] . "\n";
46    }
47    // on affiche la réponse du serveur
48    print "---Réponse du serveur [" . $response->getContent() . "]\n";
49  } catch (TypeError | RuntimeException $ex) {
50    // on affiche l'erreur
51    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
52  }
```

Commentaires

- lignes 33-37 : l'ajout de paramètres à la requête GET de la ligne 32. L'objet **[HttpClient]** s'occupera lui-même de l'encodage de l'URL ;

1.17.9.5 Le client POST

Un client HTTP envoie au serveur web la séquence de texte suivante : *entêtes HTTP, ligne vide, document*. Dans le client précédent, cette séquence était la suivante :

```
1  GET /url?paramètres HTTP/1.1
2  … autres entêtes HTTP
3  ligne vide
```

Il n'y avait pas de document. Il existe une autre façon de transmettre des paramètres, la méthode dite POST. Dans ce cas, la séquence de texte envoyée au serveur web est la suivante :

```
1  POST /url HTTP/1.1
2  … autres entêtes HTTP
3  ligne vide
4  paramètres
```

Cette fois-ci, les paramètres qui pour le client GET étaient inclus dans les entêtes HTTP, font partie, dans le client POST, du document envoyé après les entêtes.

Le script du client POST **[parameters-postclient.php]** est le suivant :

```
1  <?php
2
3  // client POST d'un serveur web
4  //
5  // gestion des erreurs
6  //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7  //ini_set("display_errors", "off");
8  //
9  // dépendances
10 require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11 use Symfony\Component\HttpClient\HttpClient;
12
13 // la configuration du client
14 const CONFIG_FILE_NAME = "config-parameters-client.json";
15
16 // on récupère la configuration
17 if (!file_exists(CONFIG_FILE_NAME)) {
18   print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
19   exit;
20 }
```

```
21  if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
22    print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
23    exit;
24  }
25
26  // on crée un client HTTP
27  $httpClient = HttpClient::create();
28  try {
29    // on prépare les paramètres
30    list($prenom, $nom, $age) = array("jean-paul", "de la hûche", 45);
31    // on fait la requête au serveur
32    $response = $httpClient->request('POST', $config['url-post'],
33      ["body" => [
34          "prenom" => $prenom,
35          "nom" => $nom,
36          "age" => $age
37    ]]);
38    // statut de la réponse
39    $statusCode = $response->getStatusCode();
40    print "---Réponse avec statut : $statusCode\n";
41    // on récupère les entêtes
42    print "---Entêtes de la réponse\n";
43    $headers = $response->getHeaders();
44    foreach ($headers as $type => $value) {
45      print "$type: " . $value[0] . "\n";
46    }
47    // on affiche la réponse du serveur
48    print "---Réponse du serveur [" . $response->getContent() . "]\n";
49  } catch (TypeError | RuntimeException $ex) {
50    // on affiche l'erreur
51    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
52  }
```

- ligne 32 : on a maintenant une requête HTTP de type POST ;
- lignes 33-37 : les paramètres du POST sont appelés le corps (body) de la requête POST : c'est le document envoyé par le client au serveur. Ici, trois paramètres sont envoyés **[nom, prenom, age]** ;
- ligne 48 : on affiche la réponse jSON du serveur ;

Les résultats de l'exécution du script client sont les suivants :

```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Mon, 03 Jun 2019 11:43:02 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: no-cache, private
7   content-length: 163
8   connection: close
9   content-type: application/json
10  ---Réponse du serveur [{"method":"POST","uri":"\/php7\/scripts-web\/05\/parameters-
    server.php","getParameters":[],"bodyParameters":{"prenom":"jean-paul","nom":"de la hûche","age":"45"}}]
```

- ligne 10 : la méthode est **[Post]** et les paramètres sont de type **[bodyParameters]**. Il n'y a pas de paramètres **[getParameters]** comme le montre l'**[uri]** ;

On peut avoir la curiosité de voir ce que reçoit le serveur lors d'une requête POST. Pour cela, nous lançons notre serveur générique **[RawTcpServer]** sur le port 100 de la machine locale à partir d'un terminal Laragon (cf paragraphe lien) :

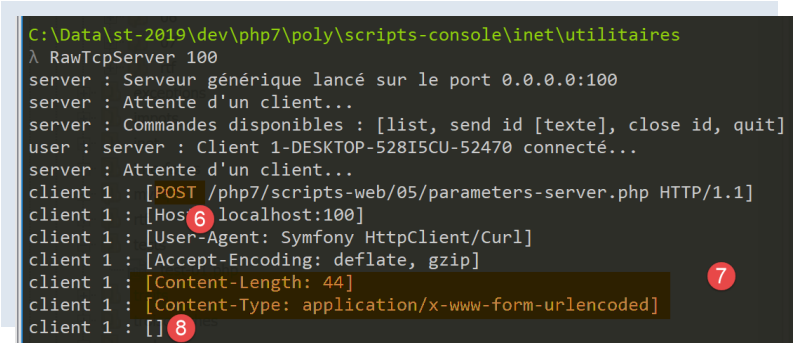
Vérifiez qu'en **[4]**, vous êtes bien dans le dossier des utilitaires.

Nous modifions le fichier jSON **[config-parameters-client.json]** qui configure le client POST :

```
1  {
2      "url-get": "http://localhost:100/php7/scripts-web/05/parameters-server.php",
3      "url-post": "http://localhost:100/php7/scripts-web/05/parameters-server.php"
4  }
```

- ligne 3 : nous avons changé le port du serveur web. Ce sera donc **[RawTcpServer]** qui sera contacté ;

Nous exécutons le client. Dans la fenêtre de **[RawTcpServer]** nous obtenons les informations suivantes :



- en **[6]**, la commande POST ;
- en **[7]** : l'entête HTTP **[Content-Length]** donne le nombre d'octets du document que va envoyer le client au serveur. L'entête HTTP **[Content-Type]** donne la nature de ce document. Le type **[application/x-www-form-urlencoded]** désigne un texte url-encodé ;
- en **[8]**, la ligne vide qui annonce la fin des entêtes HTTP et le début du document de 44 octets. Ce que ne montre pas la copie d'écran c'est le document lui-même. C'est la chaîne url-encodée des paramètres : **[prenom=jean-paul&nom=de+la+h%C3%BBche&age=45]**. Le lecteur pourra vérifier qu'elle a bien 44 caractères ;

1.17.9.6 Un client POST mixte

Dans un POST, on peut mixer les paramètres encodés dans l'URL et ceux encodés dans le document envoyé par le client après les entêtes HTTP. Voici un exemple **[parameters-mixte-postclient.php]** :

```
1  <?php
2
3  // client POST d'un serveur web
4  //
5  // gestion des erreurs
6  //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7  //ini_set("display_errors", "off");
8  //
9  // dépendances
10 require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11 use Symfony\Component\HttpClient\HttpClient;
12
13 // la configuration du client
14 const CONFIG_FILE_NAME = "config-parameters-client.json";
15
16 // on récupère la configuration
17 if (!file_exists(CONFIG_FILE_NAME)) {
18   print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
19   exit;
20 }
21 if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
22   print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
23   exit;
24 }
25
26 // on crée un client HTTP
27 $httpClient = HttpClient::create();
28 try {
29   // on prépare les paramètres
30   list($prenom, $nom, $age) = array("jean-paul", "de la hûche", 45);
31   // on fait la requête au serveur
```

```
32    $response = $httpClient->request('POST', $config['url-post'],
33      [
34        // paramètres du document (body)
35        "body" => [
36          "prenom" => $prenom,
37          "nom" => $nom,
38          "age" => $age
39        ],
40        // paramètres de l'URL (query)
41        "query" => [
42          "prenom2" => $prenom,
43          "nom2" => $nom,
44          "age2" => $age
45    ]]);
46    // statut de la réponse
47    $statusCode = $response->getStatusCode();
48    print "---Réponse avec statut : $statusCode\n";
49    // on récupère les entêtes
50    print "---Entêtes de la réponse\n";
51    $headers = $response->getHeaders();
52    foreach ($headers as $type => $value) {
53      print "$type: " . $value[0] . "\n";
54    }
55    // on affiche la réponse du serveur
56    print "---Réponse du serveur [" . $response->getContent() . "]\n";
57  } catch (TypeError | RuntimeException $ex) {
58    // on affiche l'erreur
59    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
60  }
```

Commentaires

- ligne 32 : une requête POST ;
- lignes 40-45 : les paramètres url-encodés dans l'URL ;
- lignes 35-39 : les paramètres url-encodés dans le corps (body, document) de la requête ;

A l'exécution, on obtient les résultats console suivants :

```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Mon, 03 Jun 2019 12:34:23 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: no-cache, private
7   content-length: 270
8   connection: close
9   content-type: application/json
10  ---Réponse du serveur [{"method":"POST","uri":"\/php7\/scripts-web\/05\/parameters-server.php?prenom2=jean-
    paul&nom2=de%20la%20h%C3%BBche&age2=45","getParameters":{"prenom2":"jean-paul","nom2":"de la
    hûche","age2":"45"},"bodyParameters":{"prenom":"jean-paul","nom":"de la hûche","age":"45"}}]
```

- ligne 10 : on voit que le serveur a été capable de récupérer les deux types de paramètres ;

1.17.9.7 Un client GET mixte

On essaie de faire la même chose que précédemment avec une requête GET. Le script **[parameters-mixte-get-client.php]** est le suivant :

```
1   <?php
2
3   // client POST d'un serveur web
4   //
5   // gestion des erreurs
6   //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7   //ini_set("display_errors", "off");
8   //
9   // dépendances
10  require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11  use Symfony\Component\HttpClient\HttpClient;
12
13  // la configuration du client
14  const CONFIG_FILE_NAME = "config-parameters-client.json";
15
16  // on récupère la configuration
```

```
17  if (!file_exists(CONFIG_FILE_NAME)) {
18    print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
19    exit;
20  }
21  if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
22    print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
23    exit;
24  }
25
26  // on crée un client HTTP
27  $httpClient = HttpClient::create();
28  try {
29    // on prépare les paramètres
30    list($prenom, $nom, $age) = array("jean-paul", "de la hûche", 45);
31    // on fait la requête au serveur
32    $response = $httpClient->request('GET', $config['url-post'],
33      [
34        // paramètres du document (body)
35        "body" => [
36          "prenom" => $prenom,
37          "nom" => $nom,
38          "age" => $age
39        ],
40        // paramètres de l'URL (query)
41        "query" => [
42          "prenom2" => $prenom,
43          "nom2" => $nom,
44          "age2" => $age
45    ]]);
46    // statut de la réponse
47    $statusCode = $response->getStatusCode();
48    print "---Réponse avec statut : $statusCode\n";
49    // on récupère les entêtes
50    print "---Entêtes de la réponse\n";
51    $headers = $response->getHeaders();
52    foreach ($headers as $type => $value) {
53      print "$type: " . $value[0] . "\n";
54    }
55    // on affiche la réponse du serveur
56    print "---Réponse du serveur [" . $response->getContent() . "]\n";
57  } catch (TypeError | RuntimeException $ex) {
58    // on affiche l'erreur
59    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
60  }
```

Commentaires

- ligne 32 : une requête POST ;
- lignes 40-45 : les paramètres url-encodés dans l'URL ;
- lignes 35-39 : les paramètres url-encodés dans le corps (body, document) de la requête ;

A l'exécution, on obtient les résultats console suivants :

```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Mon, 03 Jun 2019 12:41:19 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: no-cache, private
7   content-length: 217
8   connection: close
9   content-type: application/json
10  ---Réponse du serveur [{"method":"GET","uri":"\/php7\/scripts-web\/05\/parameters-server.php?prenom2=jean-
    paul&nom2=de%20la%20h%C3%BBche&age2=45","getParameters":{"prenom2":"jean-paul","nom2":"de la
    hûche","age2":"45"},"bodyParameters":[]}]
```

- ligne 10 : on constate que le serveur n'a pas reçu de paramètres url-encodés dans le document envoyé par le client. Lorsqu'on regarde les entêtes HTTP envoyés par celui-ci, on s'aperçoit qu'il a bien envoyé un document de 44 caractères mais le serveur ne l'a pas exploité ;

Finalement quelle méthode choisir pour envoyer de l'information au serveur ?

- la méthode **`[GET URL?param1=val1&param2=val2&…]`** utilise une URL paramétrée qui peut servir de lien. C'est son principal avantage : l'utilisateur peut mettre dans son marque-pages de tels liens ;

- dans d'autres applications, on peut ne pas souhaiter afficher dans une URL les paramètres envoyés au serveur. Pour des raisons de sécurité par exemple. Alors on utilisera une méthode **[POST]** et on mettra les paramètres url-encodés dans un document envoyé au serveur ;

1.17.10 Gestion des sessions web

Dans les exemples client / serveur précédents on avait le fonctionnement suivant :

- le client ouvre une connexion vers le port 80 de la machine du service web ;
- il envoie la séquence de texte : en-têtes HTTP, ligne vide, **[document] ;**
- en réponse, le serveur envoie une séquence du même type ;
- le serveur clôt la connexion vers le client ;
- le client clôt la connexion vers le serveur ;

Si le même client fait peu après une nouvelle demande au serveur web, une nouvelle connexion est créée entre le client et le serveur. Celui-ci ne peut pas savoir si le client qui se connecte est déjà venu ou si c'est une première demande. Entre deux connexions, le serveur "oublie" son client. Pour cette raison, on dit que le protocole HTTP est un protocole sans état. Il est pourtant utile que le serveur se souvienne de ses clients. Ainsi si une application est sécurisée, le client va envoyer au serveur un login et un mot de passe pour s'identifier. Si le serveur "oublie" son client entre deux connexions, celui-ci devra s'identifier à chaque nouvelle connexion, ce qui n'est pas envisageable.

Pour faire le suivi d'un client, le serveur procède de la façon suivante : lors d'une première demande d'un client, il inclut dans sa réponse un identifiant que le client doit ensuite lui renvoyer à chaque nouvelle demande. Grâce à cet identifiant, différent pour chaque client, le serveur peut reconnaître un client. Il peut alors gérer une mémoire pour ce client sous la forme d'une mémoire associée de façon unique à l'identifiant du client.

Techniquement cela se passe ainsi :

- dans la réponse à un nouveau client, le serveur inclut l'en-tête HTTP *Set-Cookie : MotClé=Identifiant*. Il ne fait cela qu'à la première demande ;
- dans ses demandes suivantes, le client va renvoyer son identifiant via l'en-tête HTTP *Cookie : MotClé=Identifiant* afin que le serveur le reconnaisse ;

On peut se demander comment le serveur fait pour savoir qu'il a affaire à un nouveau client plutôt qu'à un client déjà venu. C'est la présence de l'en-tête HTTP *Cookie* dans les en-têtes HTTP du client qui le lui indique. Pour un nouveau client, cet en-tête est absent.

L'ensemble des connexions d'un client donné est appelé une session.

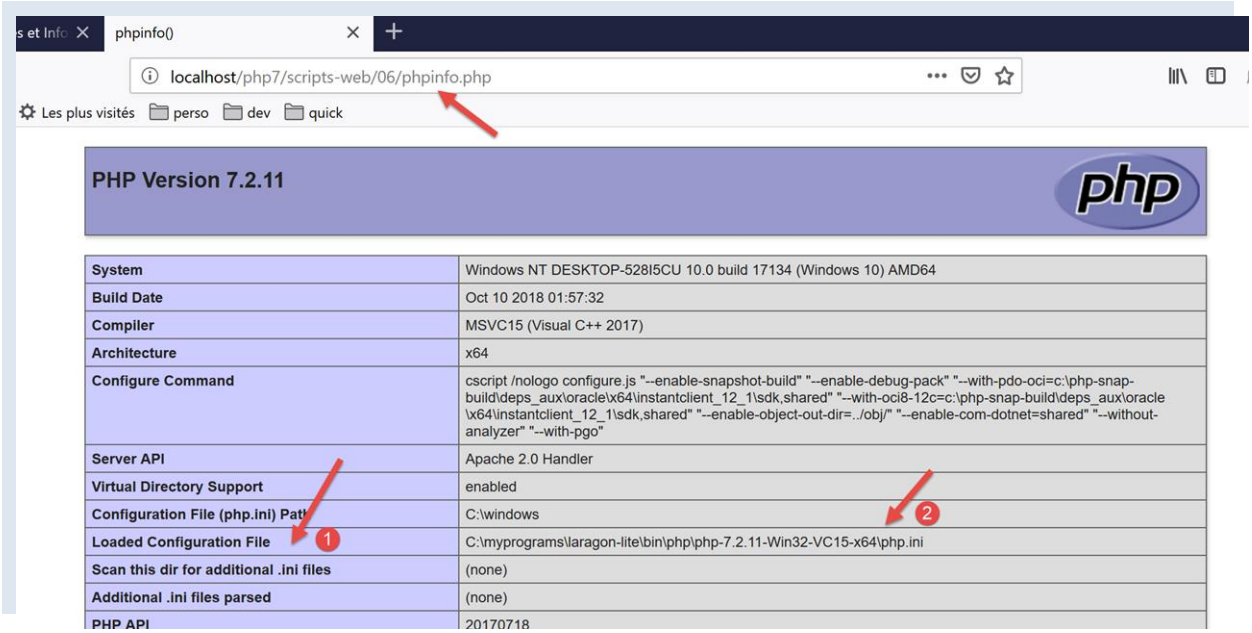
1.17.10.1 Le fichier de configuration [php.ini]

Pour que la gestion des sessions fonctionne correctement avec PHP, il faut vérifier que celui-ci est correctement configuré. Sous windows, son fichier de configuration est *php.ini*. Selon le contexte d'exécution (console, web), le fichier de configuration **[php.ini]** doit être recherché dans des dossiers différents. Pour découvrir ceux-ci, on utilisera le script suivant :

```
1   <?php
2
3   // infos PHP
4   phpinfo();
```

Ligne 4, la fonction *phpinfo* donne des informations sur l'interpréteur PHP qui exécute le script. Elle donne notamment le chemin du fichier de configuration **[php.ini]** utilisé.

Nous avons déjà utilisé ce script dans un environnement console (cf paragraphe lien). Dans un environnement web, on obtient le résultat suivant :



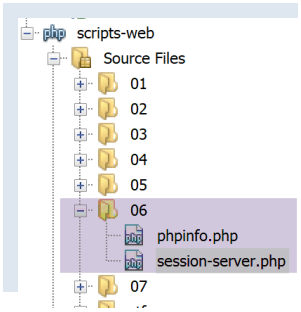
- en **[1-2]**, le fichier **[php.ini]** qui configure l'interpréteur des scripts web. On trouve dans ce fichier une section **session** :

```
1   [Session]
2   session.save_handler = files
3   session.save_path = "C:/myprograms/laragon-lite/tmp"
4   session.use_strict_mode = 0
5   session.use_cookies = 1
6   session.use_only_cookies = 1
7   session.name = PHPSESSID
8   session.auto_start = 0
9   session.cookie_lifetime = 0
10  session.cookie_path = /
11  session.cookie_domain =
12  session.cookie_httponly =
13  session.serialize_handler = php
14  session.gc_probability = 1
15  session.gc_divisor = 1000
16  session.gc_maxlifetime = 36000
17  session.referer_check =
18  session.cache_limiter = nocache
19  session.cache_expire = 180
20  session.use_trans_sid = 0
21  session.trans_sid_tags = "a=href,area=href,frame=src,form="
22  session.sid_bits_per_character = 5
```

- ligne 2 : les données d'une session client sont sauvegardées dans un fichier ;
- ligne 3 : le dossier de sauvegarde des données de session. Si ce dossier n'existe pas, aucune erreur n'est signalée et la gestion des sessions ne fonctionne pas ;
- lignes 4-6 : indiquent que l'identifiant de session est géré par les en-têtes HTTP *Set-Cookie* et *Cookie ;*
- ligne 7 : l'entête *Set-Cookie* sera de la forme *Set-Cookie : PHPSESSID=identifiant_de_session ;*
- ligne 8 : une session client n'est pas démarrée automatiquement. Le script serveur doit la demander explicitement par une instruction *session_start()* ;
- ligne 9 : le cookie de session est valable tant que le navigateur client n'a pas été fermé ;
- ligne 10 : le chemin pour lequel le cookie de session doit être renvoyé. Si **[session.cookie_path = /xxx]**, alors à chaque fois que le navigateur demande une URL de type **[/xxx/yyy/zzz]** alors il doit renvoyer le cookie. Ici le chemin **[/]** indique que le cookie doit être renvoyé pour toute URL du site ;
- ligne 13 : certains objets mis en session doivent être sérialisés pour pouvoir être stockés dans un fichier. C'est PHP qui assure cette sérialisation / désérialisation avec les fonctions **[serialize / unserialize]** ;
- ligne 16 : durée de vie au-delà de laquelle les objets de session stockés dans le fichier de sauvegarde sont considérés comme obsolètes ;
- ligne 19 : durée de vie d'une session. Au-delà de cette durée, une nouvelle session est créée et les objets sauvegardés dans la précédente session sont perdus ;

1.17.10.2 Exemple 1

1.17.10.2.1 Le serveur



La gestion de l'identifiant de session est transparent pour un service web. Cet identifiant est géré par le serveur web. Un service web a accès à la session du client via l'instruction *session_start()*. A partir de ce moment, le service web peut lire / écrire des données dans la session du client via le dictionnaire $_SESSION. Si on utilise la bibliothèque **[HttpFoundation]**, la session est disponible via l'expression **[Request→getSession]**.

Le code suivant **[session-server.php]** montre la gestion en session de trois compteurs. A chaque nouvelle requête, le script web incrémente ces compteurs et les met en session pour qu'ils puissent être récupérés lors de la requête suivante.

```
1   <?php
2   // dépendances
3   require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
4   use \Symfony\Component\HttpFoundation\Response;
5   use Symfony\Component\HttpFoundation\Request;
6   use Symfony\Component\HttpFoundation\Session\Session;
7
8   //
9   // on récupère la requête
10  $request = Request::createFromGlobals();
11  // session
12  $session = new Session();
13  $session->start();
14  // on récupère trois compteurs dans la session
15  if ($session->has("N1")) {
16    // incrémentation du compteur N1
17    $session->set("N1", (int) $session->get("N1") + 1);
18  } else {
19    // le compteur N1 n'est pas en session - on le crée
20    $session->set("N1", 0);
21  }
22  if ($session->has("N2")) {
23    // incrémentation du compteur N2
24    $session->set("N2", (int) $session->get("N2") + 1);
25  } else {
26    // le compteur N2 n'est pas en session - on le crée
27    $session->set("N2", 10);
28  }
29  if ($session->has("N3")) {
30    // incrémentation du compteur N3
31    $session->set("N3", (int) $session->get("N3") + 1);
32  } else {
33    // le compteur N3 n'est pas en session - on le crée
34    $session->set("N3", 100);
35  }
36  // on élabore la réponse
37  $response = new Response();
38  // le contenu de la réponse est du texte utf-8
39  $response->headers->set("content-type", "application/json");
40  $response->setCharset("utf-8");
41  // la réponse sera le jSON d'un tableau contenant les trois compteurs
42  $response->setContent(json_encode([
43    "N1" => $session->get("N1"),
44    "N2" => $session->get("N2"),
45    "N3" => $session->get("N3")]));
46
47  // envoi de la réponse
48  $response->send();
```

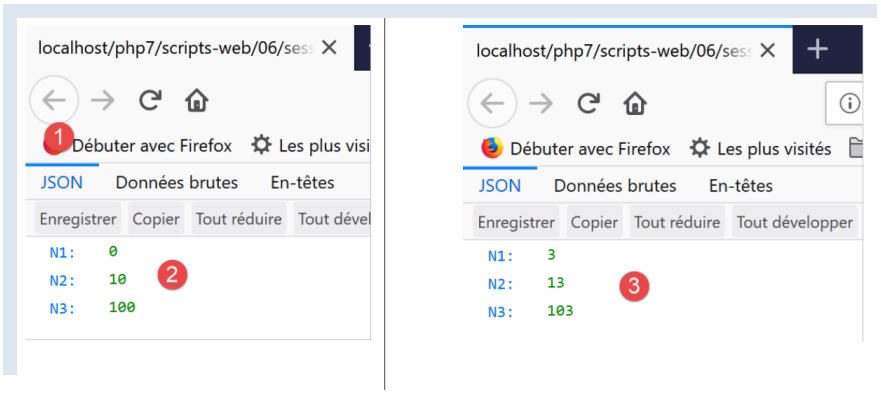
- ligne 10 : l'objet **[$request]** encapsule la totalité des informations sur la requête reçue par le script web ;
- lignes 12-13 : on crée une session et on l'active. L'objet **[Session]** encapsule les données de la session correspondant au cookie de session envoyé par le client. Si celui-ci n'a pas envoyé un tel cookie, alors il n'y aucune donnée mémorisée dans **[Session]**. Le script web va inclure dans sa 1re réponse l'entête HTTP **[Set-Cookie : PHPSESSID=xxx]**. Dans ses requêtes ultérieures, le client enverra l'entête HTTP **[Cookie : PHPSESSID=xxx]** pour indiquer la session dont il veut utiliser le contenu. Une session est la mémoire d'un client ;
- ligne 15 : on regarde si la session a une clé nommée **[N1]**. Ce sera le nom de notre 1er compteur. Si ce n'est pas le cas (ligne 20), on lui donne la valeur 0 et on le met en session. Si c'est le cas (ligne 23), on :
  - le récupère dans la session ;
  - incrémente de 1 sa valeur ;
  - le remet dans la session ;
- lignes 22-35 : on fait la même chose pour les deux autres compteurs N2 et N3 ;
- lignes 36-40 : on prépare une réponse de type **[application/json]** ;
- lignes 42-45 : la réponse sera la chaîne jSON d'un tableau contenant les trois compteurs ;
- ligne 48 : on envoie la réponse au client ;

Dans la relation client / serveur, la gestion de la session client sur le serveur dépend des deux acteurs, le client et le serveur :

- le serveur a la charge d'envoyer un identifiant à son client lors de sa première demande
- le client a la charge de renvoyer cet identifiant à chaque nouvelle demande. S'il ne le fait pas, le serveur croira que c'est un nouveau client et génèrera un nouvel identifiant pour une nouvelle session.

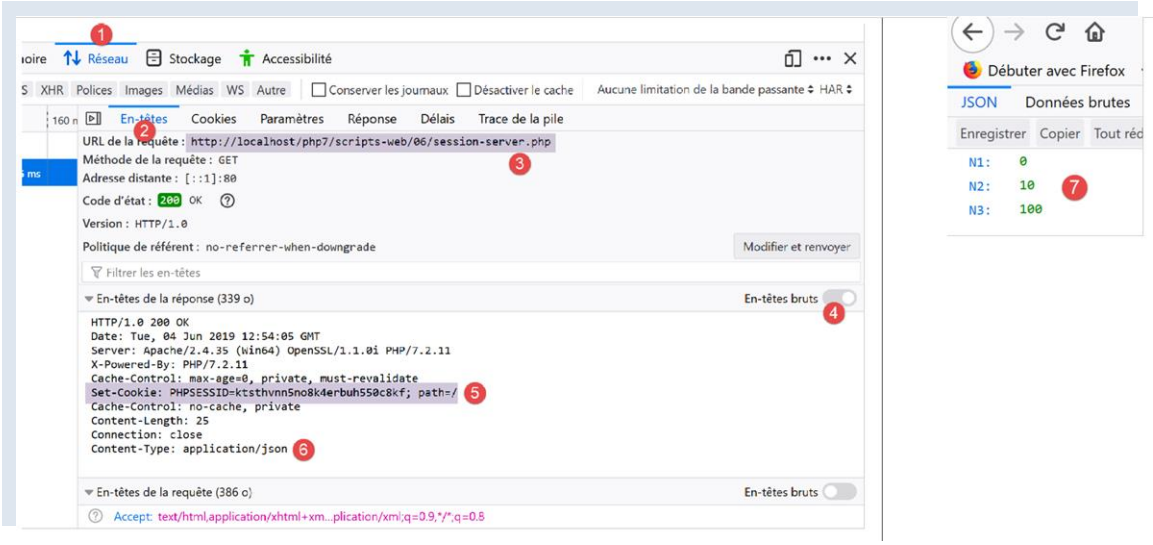
Résultats

Nous utilisons comme client, un navigateur web. Par défaut (par configuration en fait), celui-ci renvoie bien au serveur les identifiants de session que celui-ci lui envoie. Au fil des requêtes, le navigateur va recevoir les trois compteurs envoyés par le serveur et va voir leurs valeurs s'incrémenter.



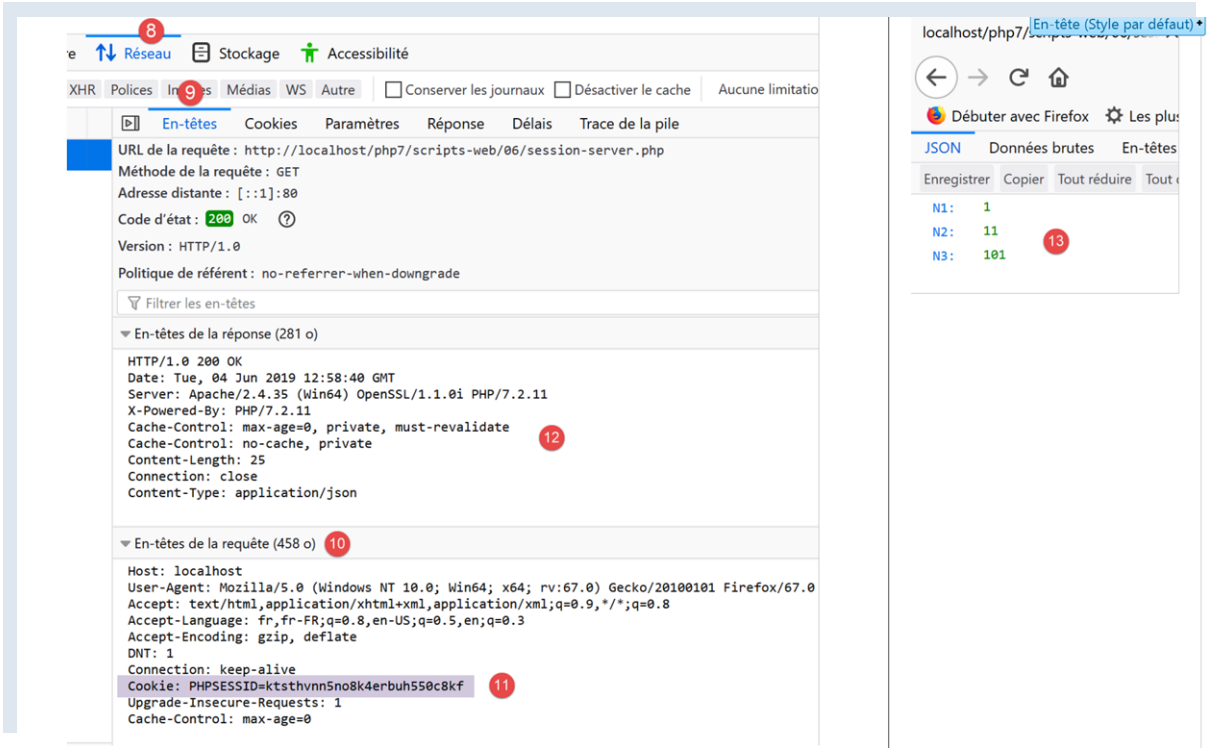
- En **[2]**, la 1re demande au service web ;
- en **[4]**, la 4e demande montre que les compteurs sont bien incrémentés. Il y a bien mémorisation des valeurs des compteurs au fil des demandes ;

Utilisons le mode développement pour voir les en-têtes HTTP échangés entre le serveur et le client. Nous fermons Firefox pour terminer la session courante avec le serveur, le rouvrons et activons le mode développement (F12). Ceci va supprimer la session courante du navigateur qui va donc en recommencer une nouvelle. Nous demandons le service **[session-server.php]** :



En **[5]**, on voit l'identifiant de session envoyé par le serveur dans sa réponse à la 1re demande du client. Il utilise l'en-tête HTTP *Set-Cookie*.

Faisons une nouvelle demande en rafraîchissant (F5) la page dans le navigateur web :



Ci-dessus on remarquera deux choses :

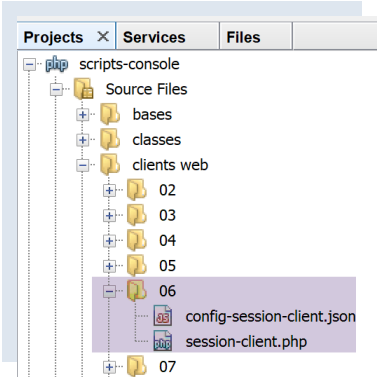
- en **[11]**, le navigateur web renvoie l'identifiant de session avec l'en-tête HTTP *Cookie*.
- En **[12]**, dans sa réponse, le service web n'inclut plus cet identifiant. C'est désormais le client qui a la charge de l'envoyer à chacune de ses demandes.

1.17.10.2.2 Le client

Nous écrivons maintenant un script client du script serveur précédent. Dans sa gestion de la session, il doit se comporter comme le navigateur web :

- dans la réponse du serveur à sa première demande, il doit trouver l'identifiant de session que le serveur lui envoie. Il sait qu'il le trouvera dans l'en-tête HTTP *Set-Cookie*.

- il doit, à chacune de ses demandes ultérieures, renvoyer au serveur l'identifiant qu'il a reçu. Il le fera avec l'en-tête HTTP *Cookie*.



Le client **[session-client]** est configuré par le fichier jSON **[config-session-client.json]** suivant :

```
{
    "url": "http://localhost/php7/scripts-web/06/session-server.php"
}
```

Le code du client **[session-client]** est le suivant :

```
<?php

// gestion d'une session
//
// gestion des erreurs
//ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
//ini_set("display_errors", "off");
//
// dépendances
require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
use Symfony\Component\HttpClient\HttpClient;

// la configuration du client
const CONFIG_FILE_NAME = "config-session-client.json";

// on récupère la configuration
if (!file_exists(CONFIG_FILE_NAME)) {
  print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
  exit;
}
if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
  print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
  exit;
}

// on crée un client HTTP
$httpClient = HttpClient::create();
try {
  // on va faire 10 requêtes
  for ($i = 0; $i < 10; $i++) {
    // on fait la requête au serveur
    if (!isset($sessionCookie)) {
      // sans session
      $response = $httpClient->request('GET', $config['url']);
    } else {
      // avec session
      $response = $httpClient->request('GET', $config['url'],
        ["headers" => ["Cookie" => $sessionCookie]]);
    }
    // statut de la réponse
    $statusCode = $response->getStatusCode();
    print "---Réponse avec statut : $statusCode\n";
    // on récupère les entêtes
    print "---Entêtes de la réponse\n";
    $headers = $response->getHeaders();
    foreach ($headers as $type => $value) {
      print "$type: " . $value[0] . "\n";
    }
```

```
49      // on récupère le cookie de session s'il existe
50      if (isset($headers["set-cookie"])) {
51        // cookie de session ?
52        foreach ($headers["set-cookie"] as $cookie) {
53          $match = [];
54          $match = preg_match("/^PHPSESSID=(.+?);/", $cookie, $champs);
55          if ($match) {
56            $sessionCookie = "PHPSESSID=" . $champs[1];
57          }
58        }
59      }
60    }
61    // on affiche la réponse jSON du serveur
62    print "---Réponse du serveur : {$response->getContent()}\n";
63  } catch (TypeError | RuntimeException $ex) {
64    // on affiche l'erreur
65    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
66  }
```

Commentaires

- ligne 27 : création du client HTTP ;
- ligne 30 : on va faire 10 fois la même requête au server **[session-server.php]** ;
- ligne 32 : la variable **[$sessionCookie]** aura pour valeur la valeur de l'entête HTTP **[Set-Cookie]** reçue par le client ;
- lignes 32-34 : si cette variable n'existe pas c'est que la session n'a pas encore démarré. On envoie la commande **[GET]** sans l'entête **[Cookie]** ;
- lignes 35-38 : sinon la session a démarré et on envoie la commande **[GET]** avec l'entête **[Cookie]**. La valeur de cet entête sera **[$sessionCookie]** ;
- ligne 50 : si l'entête **[Set-Cookie]** fait partie des entêtes HTTP reçus, alors on cherche le cookie de session ;
- ligne 52 : le serveur web peut envoyer plusieurs entêtes **[Set-Cookie]**. Le cookie de session n'est que l'un d'entre-eux. Dans notre exemple, il a la particularité d'être de la forme **[PHPSESSID=xxx;]** ;
- lignes 53-57 : on utilise une expression régulière pour trouver le cookie de session ;
- ligne 62 : une fois les 10 requêtes faites, on affiche la dernière réponse jSON du serveur ;

Résultats

L'exécution du script client provoque l'affichage suivant dans la console Netbeans :

```
1   "C:\myprograms\laragon-lite\bin\php\php-7.2.11-Win32-VC15-x64\php.exe" "C:\Data\st-
    2019\dev\php7\poly\scripts-console\clients web\06\session-client.php"
2   ---Réponse avec statut : 200
3   ---Entêtes de la réponse
4   date: Tue, 04 Jun 2019 13:41:34 GMT
5   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
6   x-powered-by: PHP/7.2.11
7   cache-control: max-age=0, private, must-revalidate
8   set-cookie: PHPSESSID=1cerjgsgdlc35e1mkenvtltmh8; path=/
9   content-length: 25
10  connection: close
11  content-type: application/json
12  ---Réponse avec statut : 200
13  ---Entêtes de la réponse
14  date: Tue, 04 Jun 2019 13:41:34 GMT
15  server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
16  x-powered-by: PHP/7.2.11
17  cache-control: max-age=0, private, must-revalidate
18  content-length: 25
19  connection: close
20  content-type: application/json
21  ---Réponse avec statut : 200
22  ---Entêtes de la réponse
23  date: Tue, 04 Jun 2019 13:41:34 GMT
24  server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
25  x-powered-by: PHP/7.2.11
26  cache-control: max-age=0, private, must-revalidate
27  content-length: 25
28  connection: close
29  content-type: application/json
30  ---Réponse avec statut : 200
31  ..........
32  ---Réponse avec statut : 200
33  ---Entêtes de la réponse
34  date: Tue, 04 Jun 2019 13:41:34 GMT
```

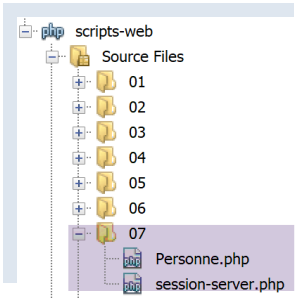
```
35  server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
36  x-powered-by: PHP/7.2.11
37  cache-control: max-age=0, private, must-revalidate
38  content-length: 25
39  connection: close
40  content-type: application/json
41  ---Réponse du serveur : {"N1":9,"N2":19,"N3":109}
```

- ligne 8 : dans sa première réponse, le serveur envoie l'identifiant de session. Dans les réponses suivantes, il ne l'envoie plus ;
- ligne 41 : les trois compteurs **[N1, N2, N3]** ont bien été incrémentés 9 fois. Lors de la requête n° 1, ils ont été mis à zéro ;

L'exemple suivant montre qu'on peut aussi sauvegarder les valeurs d'un tableau ou d'un objet dans la session.

1.17.10.3 Exemple 2

1.17.10.3.1 Le serveur



Nous allons mettre un objet **[Personne]** dans la session. La définition de cette classe est la suivante :

```
1   <?php
2
3   namespace Modèles;
4
5   class Personne implements \JsonSerializable {
6     // attributs
7     private $nom;
8     private $prénom;
9     private $âge;
10
11    // conversion d'un tableau associatif vers un objet [Personne]
12    public function setFromArray(array $assoc): Personne {
13      // on initialise l'objet courant avec le tableau associatif
14      foreach ($assoc as $attribute => $value) {
15        $this->$attribute = $value;
16      }
17      // résultat
18      return $this;
19    }
20
21    // getters et setters
22    public function getNom() {
23      return $this->nom;
24    }
25
26    public function getPrénom() {
27      return $this->prénom;
28    }
29
30    public function setNom($nom) {
31      $this->nom = $nom;
32      return $this;
33    }
34
35    public function setPrénom($prénom) {
36      $this->prénom = $prénom;
37      return $this;
38    }
39
40    public function getÂge() {
```

```
    return $this->âge;
  }

  public function setÂge($âge) {
    $this->âge = $âge;
    return $this;
  }

  // toString
  public function __toString(): string {
    return "Personne [$this->prénom, $this->nom, $this->âge]";
  }

  // implémente l'interface JsonSerializable
  public function jsonSerialize(): array {
    // on rend un tableau associatif avec pour clés les attributs de l'objet
    // ce tableau pourra ensuite être encodé en jSON
    return get_object_vars($this);
  }

  // conversion d'un jSON vers un objet [Personne]
  public static function jsonUnserialize(string $json): Personne {
    // on crée une personne à partir de la chaîne jSON
    return (new Personne())->setFromArray(json_decode($json, true));
  }

}
```

Le script serveur sera le suivant :

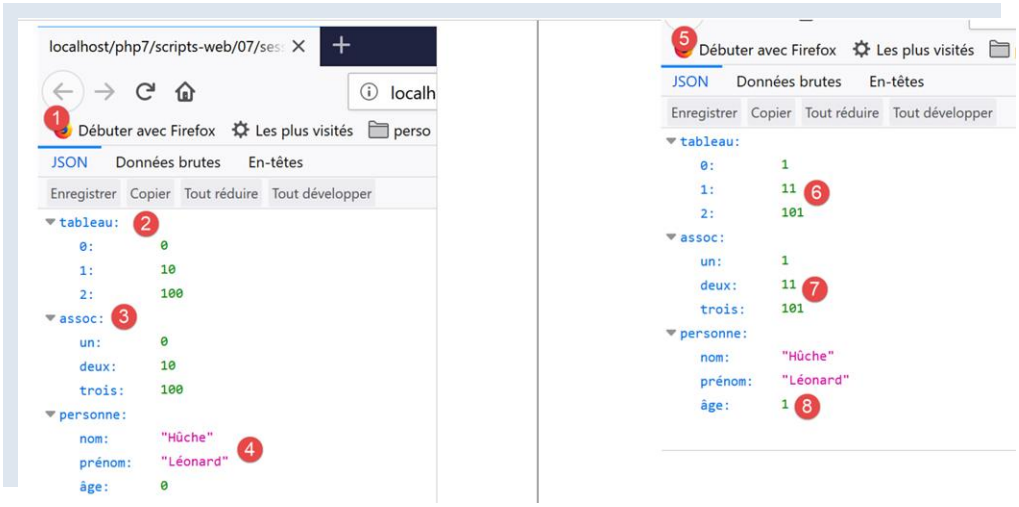
```
<?php

// dépendances
require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
use \Symfony\Component\HttpFoundation\Response;
use \Symfony\Component\HttpFoundation\Request;
use \Symfony\Component\HttpFoundation\Session\Session;
require_once __DIR__ . "/Personne.php";
use \Modèles\Personne;

//
// on récupère la requête courante
$request = Request::createFromGlobals();

// session
$session = new Session();
$session->start();

// on récupère diverses données dans la session
// tableau
if ($session->has("tableau")) {
  // le tableau est dans la session - on incrémente toutes ses valeurs
  $tableau = $session->get("tableau");
  for ($i = 0; $i < count($tableau); $i++) {
    $tableau[$i] += 1;
  }
  // on remet le tableau dans la session
  $session->set("tableau", $tableau);
} else {
  // le tableau n'est pas dans la session - on le crée
  $tableau = [0, 10, 100];
  // on le met dans la session
  $session->set("tableau", $tableau);
}
// dictionnaire
if ($session->has("assoc")) {
  // [assoc] est dans la session - on incrémente tous ses éléments
  $assoc = $session->get("assoc");
  foreach ($assoc as $key => $value) {
    $assoc[$key] = $value + 1;
  }
  // on met $assoc dans la session
  $session->set("assoc", $assoc);
} else {
  // [assoc] n'est pas dans la session - on le crée
  $assoc = ["un" => 0, "deux" => 10, "trois" => 100];
  // on met $assoc dans la session
```

```
48    $session->set("assoc", $assoc);
49  }
50  // objet Personne
51  if ($session->has("personne")) {
52    // [personne] est dans la session - on incrémente son âge
53    $personne = $session->get("personne");
54    $personne->setÂge($personne->getÂge() + 1);
55  } else {
56    // [personne] n'est pas dans la session - on le crée
57    $personne = (new Personne())->setFromArray(
58      ["prénom" => "Léonard", "nom" => "Hûche", "âge" => 0]);
59    // on met $personne dans la session
60    $session->set("personne", $personne);
61  }
62  // on élabore la réponse
63  $response = new Response();
64  // le contenu de la réponse est du jSON utf-8
65  $response->headers->set("content-type", "application/json");
66  $response->setCharset("utf-8");
67  $response->setContent(json_encode([
68    "tableau" => $tableau,
69    "assoc" => $assoc,
70    "personne" => $personne], JSON_UNESCAPED_UNICODE));
71
72  // envoi de la réponse
73  $response->send();
```

Commentaires

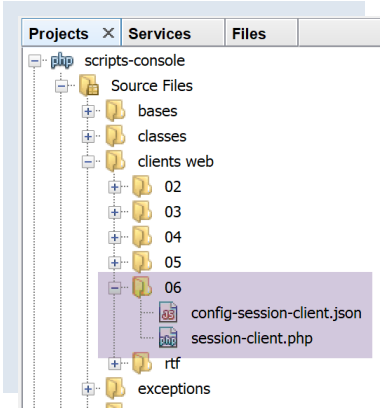
- lignes 16-17 : on récupère la session en cours et on l'active ;
- lignes 21-34 : on gère un tableau **[tableau]** mis en session. A chaque nouvelle requête, ses éléments sont incrémentés de 1 ;
- lignes 36-49 : on gère un tableau associatif **[assoc]** mis en session. A chaque nouvelle requête, ses éléments sont incrémentés de 1 ;
- lignes 51-61 : on gère un objet **[Personne]** mis en session. A chaque nouvelle requête, l'âge de cette personne est incrémentée de 1 ;
- lignes 62-73 : on envoie une réponse jSON au client : la chaîne jSON d'un tableau associatif ;

Exécutons ce script à partir de Netbeans. Les deux premières requêtes donnent les résultats suivants (F5 dans le navigateur pour la deuxième) :



- on voit qu'en **[6-8]**, tous les compteurs ont été incrémentés ;

1.17.10.3.2 Le client



Le client est le même que dans l'exemple 1 (paragraphe lien). On ne modifie que son fichier de configuration **[config-session-client]** :

```
1   {
2       "url": "http://localhost/php7/scripts-web/07/session-server.php"
3   }
```

L'exécution produit les résultats suivants :

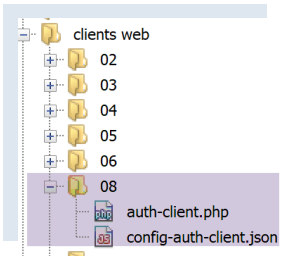
```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Tue, 04 Jun 2019 14:25:24 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: max-age=0, private, must-revalidate
7   set-cookie: PHPSESSID=qbfrj8clr20mod3eriur71mao6; path=/
8   content-length: 119
9   connection: close
10  content-type: application/json
11  ---Réponse avec statut : 200
12  ……..……..
13  ---Réponse avec statut : 200
14  ---Entêtes de la réponse
15  date: Tue, 04 Jun 2019 14:25:24 GMT
16  server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
17  x-powered-by: PHP/7.2.11
18  cache-control: max-age=0, private, must-revalidate
19  content-length: 119
20  connection: close
21  content-type: application/json
22  ---Réponse du serveur :
    {"tableau":[9,19,109],"assoc":{"un":9,"deux":19,"trois":109},"personne":{"nom":"Hûche","prénom":"Léonard","
    âge":9}}
```

- en ligne **[22]**, on voit que tous les compteurs ont été incrémentés ;

1.17.11 Authentification

Nous nous intéressons maintenant aux services web destinés à certains utilisateurs seulement. Le client doit alors s'identifier auprès du service web avant d'avoir sa réponse.

1.17.11.1 Le client



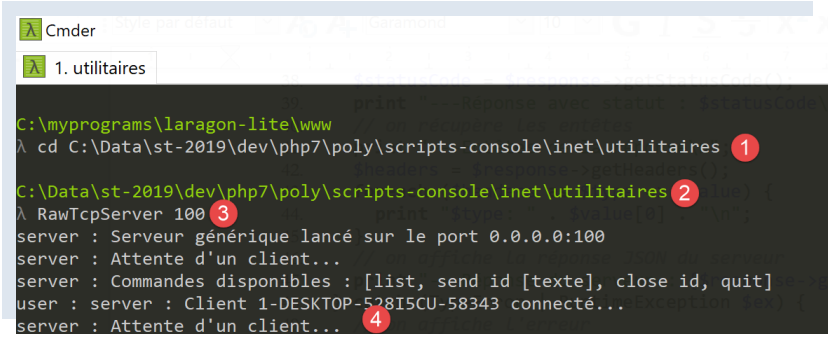
Le code du client **[auth-client.php]** est le suivant :

```
1   <?php
2
3   // gestion d'une session
4   //
5   // gestion des erreurs
6   //ini_set("error_reporting", E_ALL & ~ E_WARNING & ~E_DEPRECATED & ~E_NOTICE);
7   //ini_set("display_errors", "off");
8   //
9   // dépendances
10  require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
11  use Symfony\Component\HttpClient\HttpClient;
12
13  // la configuration du client
14  const CONFIG_FILE_NAME = "config-auth-client.json";
15
16  // on récupère la configuration
17  if (!file_exists(CONFIG_FILE_NAME)) {
18    print "Le fichier de configuration [" . CONFIG_FILE_NAME . "] n'existe pas\n";
19    exit;
20  }
21  if (!$config = \json_decode(\file_get_contents(CONFIG_FILE_NAME), true)) {
22    print "Erreur lors de l'exploitation du fichier de configuration jSON [" . CONFIG_FILE_NAME . "]\n";
23    exit;
24  }
25
26  // on crée un client HTTP
27  $httpClient = HttpClient::create([
28      'auth_basic' => ['admin', 'admin'],
29      // "verify_peer" => false,
30      // "verify_host" => false
31    ]);
32
33
34  try {
35    // on fait la requête au serveur
36    $response = $httpClient->request('GET', $config['url']);
37    // statut de la réponse
38    $statusCode = $response->getStatusCode();
39    print "---Réponse avec statut : $statusCode\n";
40    // on récupère les entêtes
41    print "---Entêtes de la réponse\n";
42    $headers = $response->getHeaders();
43    foreach ($headers as $type => $value) {
44      print "$type: " . $value[0] . "\n";
45    }
46    // on affiche la réponse jSON du serveur
47    print "---Réponse du serveur : {$response->getContent()}\n";
48  } catch (TypeError | RuntimeException $ex) {
49    // on affiche l'erreur
50    print "Erreur de communication avec le serveur : " . $ex->getMessage() . "\n";
51  }
```

Commentaires

- lignes 27-31 : on a passé un paramètre à la méthode statique **[HttpClient::create]**, un tableau associatif ;
- ligne 28 : la clé **[auth_basic]** a pour valeur un tableau à deux éléments **[user, password]**. C'est avec ces éléments que le client va s'authentifier auprès du service web. La clé **[auth_basic]** désigne un type d'authentification appelé **[Autorization Basic]** du nom de l'entête HTTP que va émettre le client. Il existe d'autres types d'authentification ;
- en-dehors de ce code, le client est identique aux précédents ;

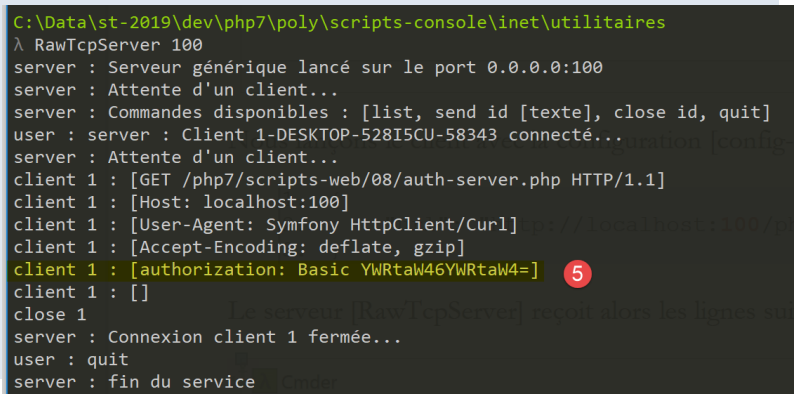
Pour voir les entêtes HTTP envoyés par le client, nous allons le connecter au serveur TCP générique **[RawTcpServer]** comme nous l'avons fait déjà de nombreuses fois :



Nous lançons le client avec la configuration **[config-auth-client.json]** suivante :

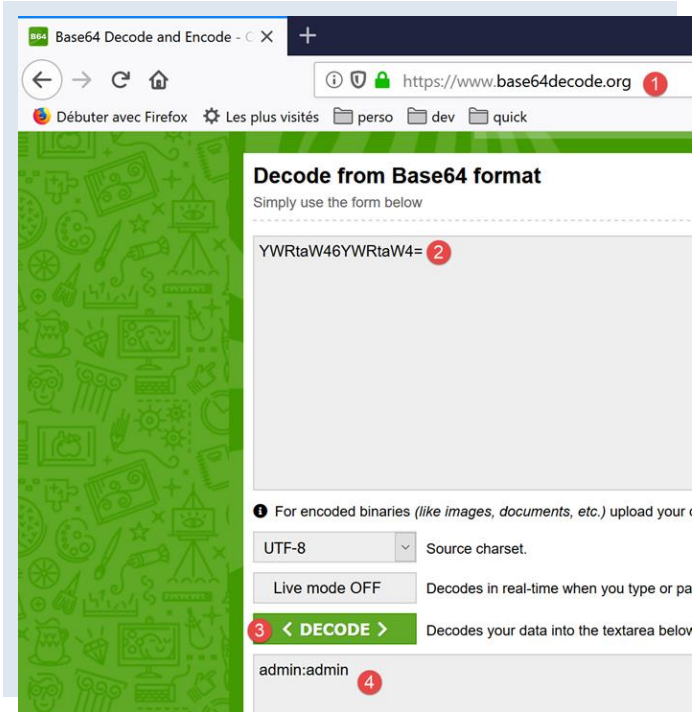
```
1   {
2       "url": "http://localhost:100/php7/scripts-web/08/auth-server.php"
3   }
```

Le serveur **[RawTcpServer]** reçoit alors les lignes suivantes :

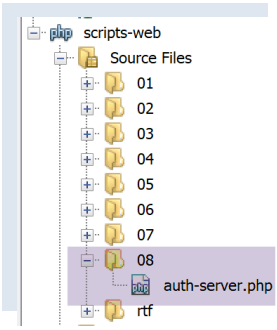


- en **[5]**, on voit l'entête **[Autorization : Basic XXX]** envoyé par le client. La chaîne XXX est la chaîne **[user:password]** codé en Base64 ;

Pour vous en assurer, vous pouvez décoder la chaîne reçue sur le site **[https://www.base64decode.org/]** :



1.17.11.2 Le serveur



Le serveur **[auth-server.php]** est le suivant :

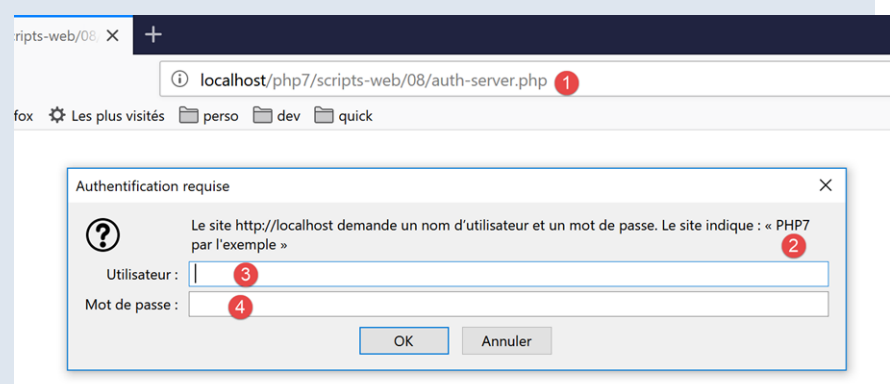
```
1   <?php
2
3   // dépendances
4   require_once 'C:/myprograms/laragon-lite/www/vendor/autoload.php';
5   use \Symfony\Component\HttpFoundation\Response;
6   use \Symfony\Component\HttpFoundation\Request;
7
8   // utilisateurs autorisés
9   $users = ["admin" => "admin"];
10  //
11  // on récupère la requête courante
12  $request = Request::createFromGlobals();
13  // authentification
14  $requestUser = $request->headers->get('php-auth-user');
15  $requestPassword = $request->headers->get('php-auth-pw');
16  // l'utilisateur existe-t-il ?
17  $trouvé = array_key_exists($requestUser, $users) && $users[$requestUser] === $requestPassword;
18  // préparation de la réponse
19  $response = new Response();
20  // on fixe le code de statut de la réponse
21  if (!$trouvé) {
22    // pas trouvé - code 401
23    $response->setStatusCode(Response::HTTP_UNAUTHORIZED);
24    $response->headers->add(["WWW-Authenticate"=> "Basic realm=".utf8_decode("\"PHP7 par l'exemple\"")]);
```

```
25  } else {
26    // trouvé - code 200
27    $response->setStatusCode(Response::HTTP_OK);
28  }
29  // la réponse n'a pas de contenu, seulement des entêtes HTTP
30  $response->send();
```

Commentaires

- ligne 9 : les utilisateurs autorisés, ici un seul de login **[admin]** avec le mot de passe **[admin]** ;
- ligne 14 : l'identifiant de l'utilisateur est récupéré dans l'entête **[PHP-AUTH-USER]**. Ce n'est pas un entête que le client a envoyé, mais un entête que le PHP du serveur a construit ;
- ligne 15 : le mot de passe de l'utilisateur est récupéré dans l'entête **[PHP-AUTH-PW]**, un entête construit par PHP ;
- ligne 17 : on cherche l'utilisateur qui veut se connecter dans la liste des utilisateurs autorisés ;
- lignes 23-24 : si l'utilisateur n'a pas été reconnu, on envoie au client
  - ligne 23 : le code **[401 Unauthorized]** ;
  - ligne 24 : un entête **[WWW-Authenticate: Basic realm="quelque chose"]**. La plupart des navigateurs reconnaissent cet entête et vont faire apparaître une fenêtre d'authentification invitant l'utilisateur à s'authentifier. Les entêtes HTTP doivent être codés en ISO 8859-1. Les textes Netbeans sont eux codés en UTF-8. La fonction **[utf8_decode]** assure la conversion UTF-8 vers ISO 8859-1. Ici, elle n'était pas nécessaire car les caractères de la chaîne **[PHP7 par l'exemple]** sont les mêmes en UTF-8 et ISO 8859-1. La fonction n'est là que pour un rappel du codage utilisé par les entêtes HTTP ;
- ligne 25 : si l'utilisateur a été reconnu, on envoie au client le code **[200 OK]** ;

Demandons l'URL **[auth-server.php]** avec un navigateur :



On voit que le navigateur fait afficher une fenêtre d'authentification. En **[2]**, on voit la valeur de l'entête **[WWW-Authenticate]** envoyé par le serveur. Si on regarde les entêtes HTTP reçus par le navigateur, on trouve la chose suivante :

```
1   HTTP/1.0 401 Unauthorized
2   Date: Fri, 07 Jun 2019 09:11:23 GMT
3   Server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
4   X-Powered-By: PHP/7.2.11
5   Cache-Control: no-cache, private
6   WWW-Authenticate: Basic realm="PHP7 par l'exemple"
7   Content-Length: 0
8   Connection: close
9   Content-Type: text/html; charset=UTF-8
```

- ligne 1 : le code **[401 Unauthorized]** de la réponse ;
- ligne 6 : l'entête HTTP **[WWW-Authenticate]** ;
- ligne 7 : le corps de la réponse est vide ;

Si en **[3-4]**, on tape **[admin]** deux fois, la réponse du serveur est alors la suivante :

```
1   HTTP/1.0 200 OK
2   Date: Fri, 07 Jun 2019 09:21:00 GMT
3   Server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
4   X-Powered-By: PHP/7.2.11
5   Cache-Control: no-cache, private
6   Content-Length: 0
7   Connection: close
8   Content-Type: text/html; charset=UTF-8
```

- ligne 1 : le code 200 OK de la réponse ;
- ligne 6 : le corps de la réponse est vide ;

Si en **[3-4]**, on tape des identifiants erronés, le navigateur **[Firefox]** utilisé pour les tests, représente indéfiniment le fenêtre d'authentification jusqu'à ce que les identifiants corrects soient tapés. A chaque fois un aller-retour avec le serveur a lieu avec toujours la même réponse qui déclenche la fenêtre d'authentification du navigateur.

Exécutons le client **[auth-client.php]** avec un utilisateur non autorisé. La réponse du serveur est la suivante :

```
1   ---Réponse avec statut : 401
2   ---Entêtes de la réponse
3   Erreur de communication avec le serveur : HTTP/1.0 401 Unauthorized returned for
    "https://localhost/php7/scripts-web/08/auth-server.php".
```

- En **[1]**, le client a bien reçu un code 401 ;
- en **[3]**, une exception a été lancée dans le client. C'est le client Symfony **[HttpClient]** qui l'a lancée : il lance une exception lorsque que le code de statut de la réponse HTTP indique qu'il y a eu erreur côté serveur, et que le client essaie de lire les entêtes ou le contenu de la réponse du serveur. Le message de la ligne 3 nous permet de voir que le serveur a répondu par **[HTTP/1.0 401 Unauthorized]** pour indiquer que l'utilisateur n'avait pas été reconnu ;

Exécutons maintenant le client **[auth-client.php]** avec l'utilisateur autorisé **['admin','admin']**. La réponse du serveur est alors la suivante :

```
1   ---Réponse avec statut : 200
2   ---Entêtes de la réponse
3   date: Wed, 05 Jun 2019 10:11:02 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: no-cache, private
7   content-length: 0
8   connection: close
9   content-type: text/html; charset=UTF-8
10  ---Réponse du serveur :
11
```

- ligne 1 : le serveur a répondu **[HTTP/1. 200 OK]** ;
- ligne 7 : la réponse n'a pas de contenu (0 octet) ;

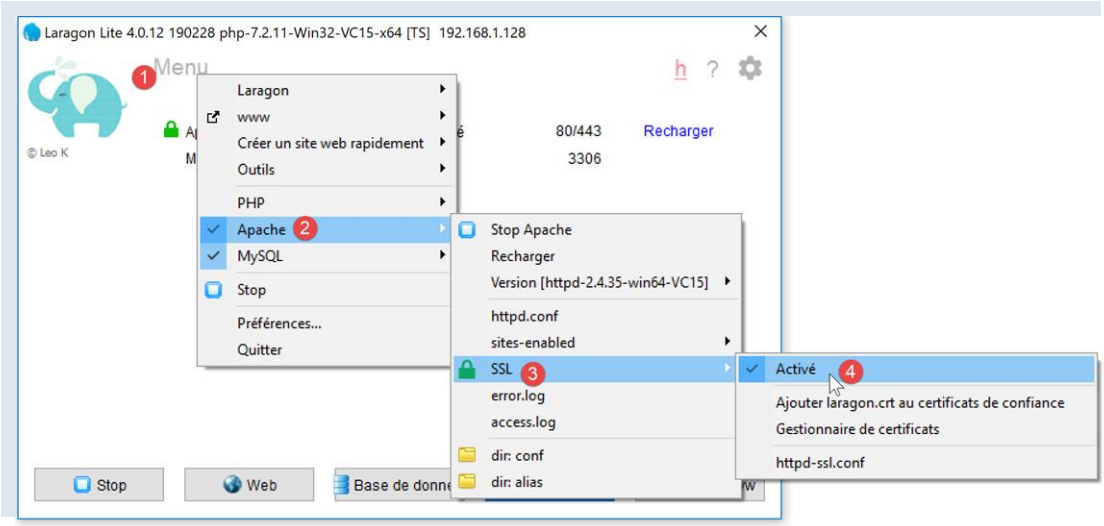
1.17.11.3 Sécuriser la connexion client / serveur

Nous avons vu que pour s'authentifier auprès du serveur, le client envoyait l'entête :

```
authorization: Basic YWRtaW46YWRtaW4=
```

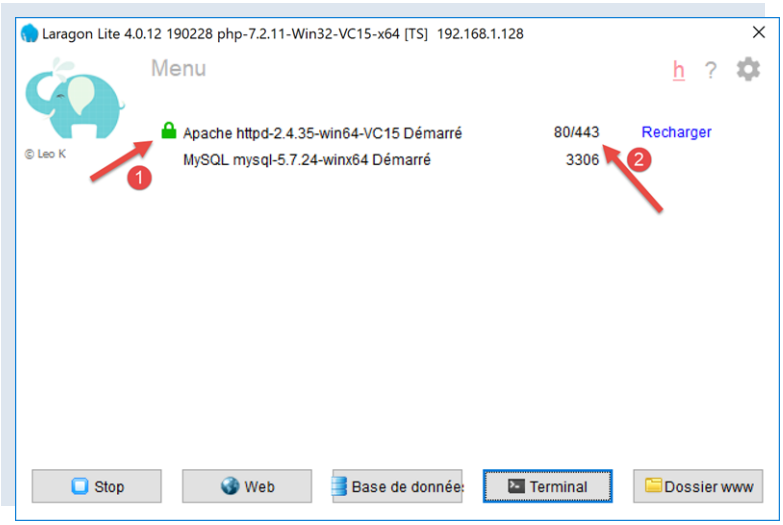
Si cette ligne est interceptée par un programme espion, celui-ci pourra aisément retrouver les identifiants **[login, mot de passe]** encodés en base 64 dans la chaîne **[YWRtaW46YWRtaW4=]**. Pour cette raison, il faut que l'authentification ait lieu dans une connexion sécurisée entre le client et le serveur. Les URL sécurisées utilisent le protocole **[HTTPS]** au lieu du protocole HTTP. Le protocole **[HTTPS]** est le protocole HTTP au sein d'une connexion client / serveur sécurisée. Les URL sécurisées sont de la forme **[https://chemin_document]**.

Tous les serveurs web n'acceptent pas les URL de cette forme. Il faut les modifier pour qu'ils soient sécurisés. Le serveur Apache de Laragon est un serveur sécurisé mais le protocole HTTPS n'est pas actif par défaut. Il faut l'activer dans le menu de Laragon :



- en **[4]**, il faut activer le chiffrement SSL du serveur Apache ;

Ceci fait, le serveur Apache est automatiquement redémarré :



- en **[1]**, un cadenas vert apparaît : c'est le signe que le protocole HTTPS a été activé ;
- en **[2]**, un nouveau port de service apparaît, le port 443 ici. C'est le port de service du protocole sécurisé HTTPS ;

Maintenant que nous avons un serveur sécurisé, modifions le fichier de configuration **[config-auth-client.json]** du client de la façon suivante :

```
1   {
2       "url": "https://localhost:443/php7/scripts-web/08/auth-server.php"
3   }
```

En **[2]**, le protocole est devenu **[https]** et le port **[443]**.

Maintenant exécutons le client **[auth-client.php]** avec l'utilisateur autorisé **[admin, admin]**. Les résultats console sont les suivants :

```
Erreur de communication avec le serveur : Peer certificate cannot be authenticated with given CA
certificates for"https://localhost/php7/scripts-web/08/auth-server.php".
```

Le client Symfony **[HttpClient]** a lancé une exception car le serveur lui a envoyé un certificat de confiance que **[HttpClient]** n'a pas accepté. La communication SSL se fait avec des certificats de confiance certifiés par des organismes officiels. Lorsqu'on a activé le protocole HTTPS sur le serveur Apache de Laragon, un certificat autosigné a été généré pour le serveur Apache. Un certificat autosigné est un certificat non validé par un organisme officiel. Le client Symfony **[HttpClient]** a refusé ce certificat autosigné.

Il est possible de demander à **[HttpClient]** de ne pas vérifier la validité du certificat envoyé par le serveur. Cela se fait avec des options dans la méthode **[HttpClient::create]** :

```
1   // on crée un client HTTP
2   $httpClient = HttpClient::create([
3       'auth_basic' => ['admin', 'admin'],
4       "verify_peer" => false
5     ]);
```

La ligne 4 demande à ce que le certificat du serveur ne soit pas vérifié. Nous avions déjà rencontré ce problème dans le script **[http-02.php]** du paragraphe lien. Ce script utilisait la bibliothèque **[libcurl]** pour se connecter à des sites HTTP et HTTPS. On avait alors utilisé la configuration suivante pour cette bibliothèque :

```
1   // Initialisation d'une session cURL
2     $curl = curl_init($url);
3     if ($curl === FALSE) {
4       // il y a eu une erreur
5       return "Erreur lors de l'initialisation de la session cURL pour le site [$site]";
6     }
7     // options de curl
8     $options = [
9       // mode verbose
10      CURLOPT_VERBOSE => true,
11      // nouvelle connexion - pas de cache
12      CURLOPT_FRESH_CONNECT => true,
13      // timeout de la requête (en secondes)
14      CURLOPT_TIMEOUT => $timeout,
15      CURLOPT_CONNECTTIMEOUT => $timeout,
16      // ne pas vérifier la validité des certificats SSL
17      CURLOPT_SSL_VERIFYPEER => false,
18      // suivre les redirections
19      CURLOPT_FOLLOWLOCATION => true,
20      // récupération du document demandé sous la forme d'une chaîne de caractères
21      CURLOPT_RETURNTRANSFER => true
22    ];
23
24    // paramétrage de curl
25  curl_setopt_array($curl, $options);
```

Ligne 17, la constante **[CURLOPT_SSL_VERIFYPEER]** contrôle la vérification ou non du certificat envoyé par le serveur. Le client **[HttpClient]** est en fait un client **[curl]** lorsque l'extension **[curl]** est activée dans la configuration de PHP comme c'est le cas ici. La classe instanciée par **[HttpClient::create]** est alors la classe **[CurlHttpClient]**. Les constantes de **[curl]** sont disponibles dans cette classe mais sous d'autres noms :

```
1   $curlopts = [
2               CURLOPT_URL => $url,
3               CURLOPT_USERAGENT => 'Symfony HttpClient/Curl',
4               CURLOPT_TCP_NODELAY => true,
5               CURLOPT_PROTOCOLS => CURLPROTO_HTTP | CURLPROTO_HTTPS,
6               CURLOPT_REDIR_PROTOCOLS => CURLPROTO_HTTP | CURLPROTO_HTTPS,
7               CURLOPT_FOLLOWLOCATION => true,
8               CURLOPT_MAXREDIRS => 0 < $options['max_redirects'] ? $options['max_redirects'] : 0,
9               CURLOPT_COOKIEFILE => '', // Keep track of cookies during redirects
10              CURLOPT_CONNECTTIMEOUT_MS => 1000 * $options['timeout'],
11              CURLOPT_PROXY => $options['proxy'],
12              CURLOPT_NOPROXY => $options['no_proxy'] ?? $_SERVER['no_proxy'] ?? $_SERVER['NO_PROXY'] ?? '',
13              CURLOPT_SSL_VERIFYPEER => $options['verify_peer'],
14              CURLOPT_SSL_VERIFYHOST => $options['verify_host'] ? 2 : 0,
15              CURLOPT_CAINFO => $options['cafile'],
16              CURLOPT_CAPATH => $options['capath'],
17              CURLOPT_SSL_CIPHER_LIST => $options['ciphers'],
18              CURLOPT_SSLCERT => $options['local_cert'],
19              CURLOPT_SSLKEY => $options['local_pk'],
20              CURLOPT_KEYPASSWD => $options['passphrase'],
21              CURLOPT_CERTINFO => $options['capture_peer_cert_chain'],
22          ];
```

Nous avons surligné en jaune, les constantes utilisées par **[CurlHttpClient]**.

Si maintenant, nous exécutons le client **[auth-client]** avec l'utilisateur **[admin, admin]** nous obtenons le résultat suivant :

```
1   ---Réponse avec statut : 200
```

```
2   ---Entêtes de la réponse
3   date: Wed, 05 Jun 2019 10:44:37 GMT
4   server: Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11
5   x-powered-by: PHP/7.2.11
6   cache-control: no-cache, private
7   content-length: 0
8   connection: close
9   content-type: text/html; charset=UTF-8
10  ---Réponse du serveur :
```

L'utilisateur a bien été reconnu. Si nous exécutons le client **[auth-client]** avec un utilisateur autre que **[admin, admin]**, nous obtenons le résultat suivant :

```
1   ---Réponse avec statut : 403
2   ---Entêtes de la réponse
3   Erreur de communication avec le serveur : HTTP/1.0 403 Forbidden returned for
    "https://localhost/php7/scripts-web/08/auth-server.php".
```

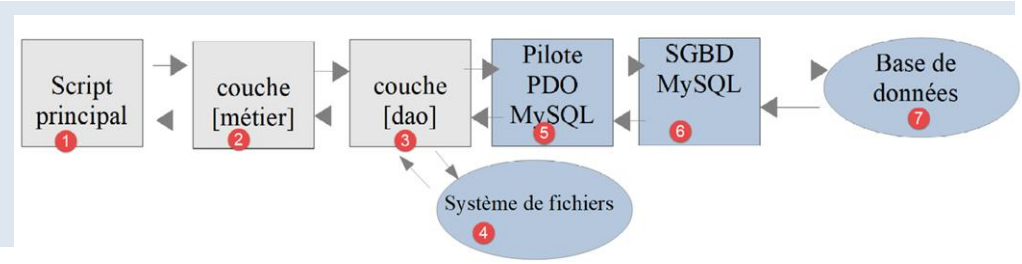
Désormais, nous savons nous authentifier auprès d'un serveur sécurisé.

1.18 Exercice d'application – version 8

Nous allons reprendre l'application exemple – version 5 (paragraphe lien) et allons en faire une application client / serveur.

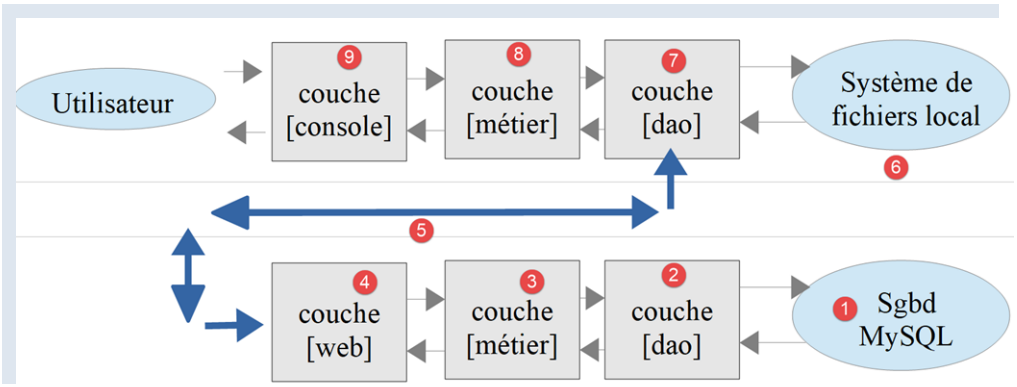
1.18.1 Introduction

L'architecture de la version 5 était la suivante :



- la couche appelée **[dao]** (Data Access Objects) s'occupe des échanges avec la base de données MySQL et le système de fihiers local ;
- la couche appelée **[métier]** fait le calcul de l'impôt ;
- le script principal est le chef d'orchestre : il instancie les couches **[dao]** et **[métier]** puis dialogue avec la couche **[métier]** pour faire ce qu'il y a faire ;

Nous allons migrer cette architecture vers l'architecture client / serveur suivante :

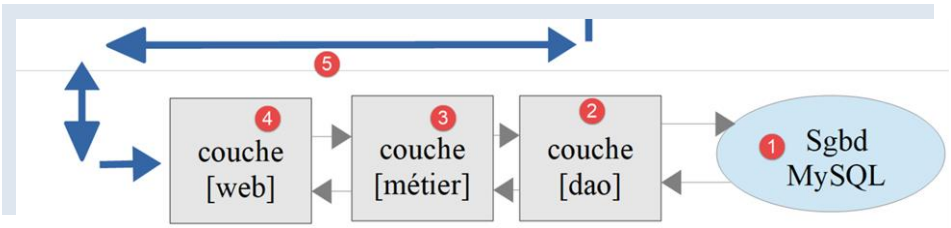


- en **[2]**, nous reprendrons la couche **[dao]** de la version 5 en lui enlevant les méthodes d'accès au système de fichiers local. Ces méthodes migreront dans la couche **[dao]** du client **[6, 7]** ;
- en **[3]**, la couche **[métier]** restera celle de la version 5 sans ses méthodes **[executeBatchImpôts, saveResults]** qui migrent dans la couche **[dao]** **[7]** du client ;

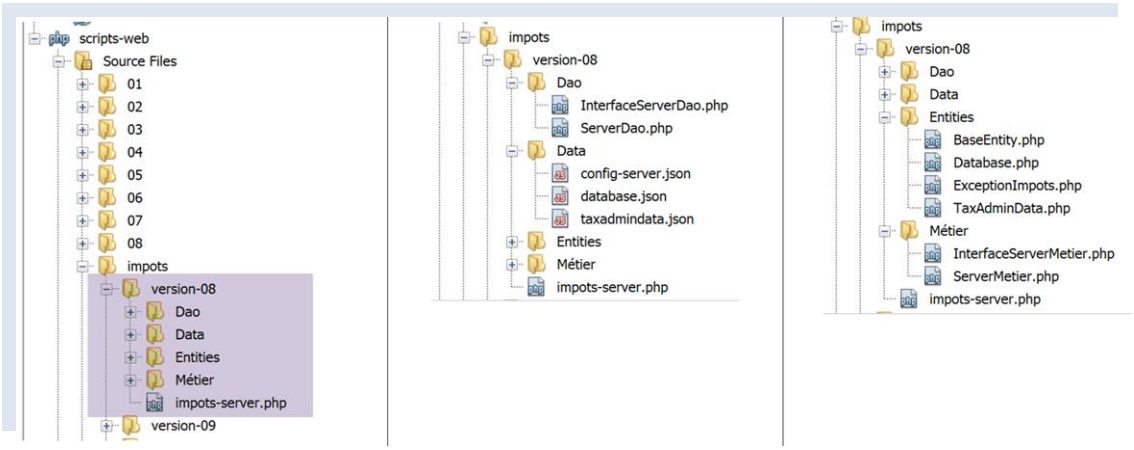
- en **[4]**, le script serveur est à écrire : il aura à :
  - créer les couches **[métier]** et **[dao]** **[3, 2]** ;
  - dialoguer avec le script client **[5, 7]** ;
- en **[7]**, la couche **[dao]** du client est à écrire :
  - elle sera un client HTTP du script serveur **[4, 5]** ;
  - elle reprendra les méthodes d'accès au système de fichiers local de la couche **[dao]** de la version 5 ;
- en **[8]**, la couche **[métier]** du client respectera l'interface **[InterfaceMetier]** de la version 5. Son implémentation sera cependant différente. Dans la version 5, la couche **[métier]** faisait le calcul de l'impôt. Ici, c'est la couche **[métier]** du serveur qui fait ce calcul. La couche **[métier]** fera donc appel à la couche **[dao]** **[7]**, pour dialoguer avec le serveur et lui demander de calculer l'impôt ;
- en **[9]**, le script console aura à instancier les couches **[dao, métier]** du client et à lancer l'exécution de celui-ci ;

1.18.2 Le serveur

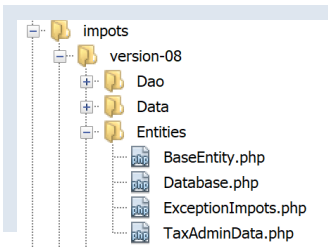
Nous nous intéressons à la partie serveur de l'application.



Cette architecture sera implémentée par les scripts suivants :

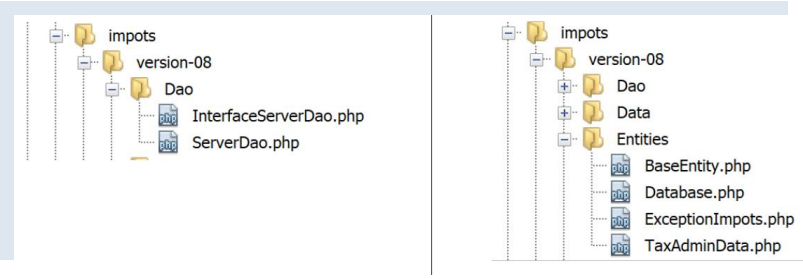


1.18.2.1 Les entités échangées entre les couches



Les entités échangées entre les couches sont celles de la version 5 décrites au paragraphe lien.

1.18.2.2 La couche [dao]



La couche **[dao]** implémente l'interface **[InterfaceServerDao]** suivante :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   interface InterfaceServerDao {
7
8     // lecture des données de l'administration fiscale
9     public function getTaxAdminData(): TaxAdminData;
10  }
```

- ligne 9 : la méthode **[getTaxAdminData]** va chercher les données de l'administration fiscale dans une base de données ;

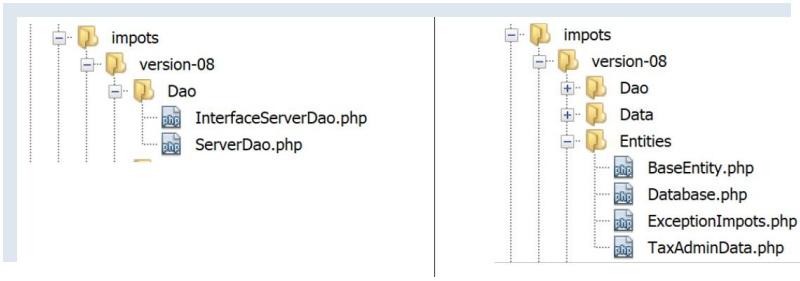
L'interface **[InterfaceServerDao]** est implémentée par la classe **[ServerDao]** suivante :

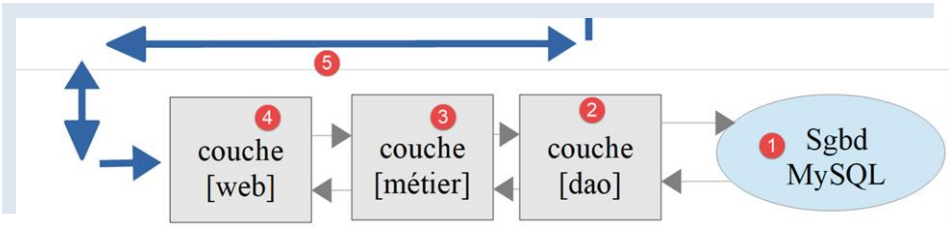
```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   // définition d'une classe ImpotsWithDataInDatabase
7   class ServerDao implements InterfaceServerDao {
8     // l'objet de type TaxAdminData qui contient les données des tranches d'impôts
9     private $taxAdminData;
10    // l'objet de type [Database] contennat les caractéristiques de la BD
11    private $database;
12
13    // constructeur
14    public function __construct(string $databaseFilename) {
15      // on mémorise la configuration JSON de la bd
16      $this->database = (new Database())->setFromJsonFile($databaseFilename);
17      // on prépare l'attribut
18      $this->taxAdminData = new TaxAdminData();
19      try {
20        // on ouvre la connexion à la base de données
21        $connexion = new \PDO($this->database->getDsn(), $this->database->getId(), $this->database->getPwd());
22        // on veut qu'à chaque erreur de SGBD, une exception soit lancée
23        $connexion->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
24        // on démarre une transaction
25        $connexion->beginTransaction();
26        // on remplit la table des tranches d'impôt
27        $this->getTranches($connexion);
28        // on remplit la table des constantes
29        $this->getConstantes($connexion);
30        // on termine la transaction sur un succès
31        $connexion->commit();
32      } catch (\PDOException $ex) {
33        // y-a-t-il une transaction en cours ?
34        if (isset($connexion) && $connexion->inTransaction()) {
35          // on termine la transaction sur un échec
36          $connexion->rollBack();
37        }
38        // on remonte l'exception au code appelant
39        throw new ExceptionImpots($ex->getMessage());
40      } finally {
41        // on ferme la connexion
42        $connexion = NULL;
43      }
44    }
45
```

```
46    // lecture des données de la base
47    private function getTranches($connexion): void {
48      …
49    }
50
51    // lecture de la table des constantes
52    private function getConstantes($connexion): void {
53      …
54    }
55
56    // retourne les données permettant le calcul de l'impôt
57    public function getTaxAdminData(): TaxAdminData {
58      return $this->taxAdminData;
59    }
60
61  }
```

Ce code a été présenté au paragraphe lien.

1.18.2.3 La couche [métier]





La couche **[métier]** implémente l'interface **[InterfaceServerMetier]** suivante :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   interface InterfaceServerMetier {
7
8     // calcul des impôts d'un contribuable
9     public function calculerImpot(string $marié, int $enfants, int $salaire): array;
10  }
```

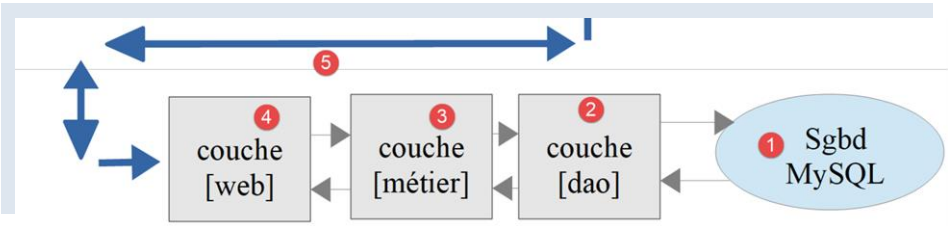
L'interface **[InterfaceServerMetier]** est implémentée par la classe **[ServerMetier]** suivante :

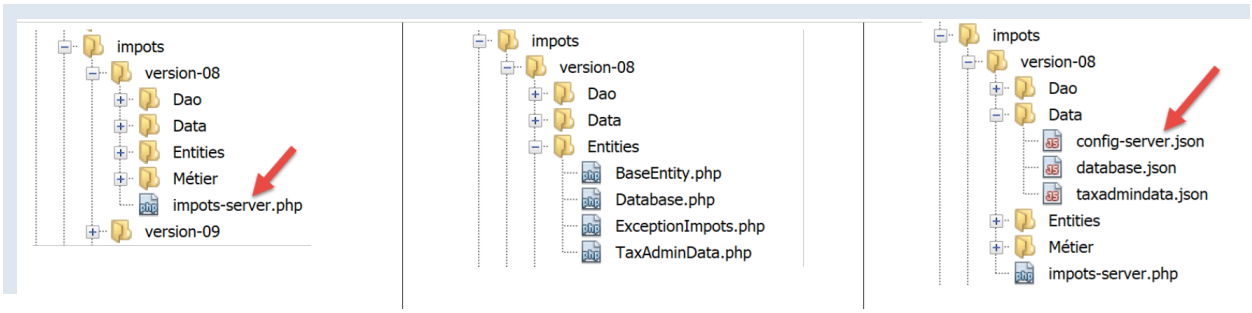
```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   class ServerMetier implements InterfaceServerMetier {
7     // couche Dao
8     private $dao;
9     // données administration fiscale
10    private $taxAdminData;
11
12    //---------------------------------------------
13    // setter couche [dao]
14    public function setDao(InterfaceServerDao $dao) {
15      $this->dao = $dao;
```

```
16      return $this;
17    }
18
19    public function __construct(InterfaceServerDao $dao) {
20      // on mémorise une référence sur la couche [dao]
21      $this->dao = $dao;
22      // on récupère les données permettant le calcul de l'impôt
23      // la méthode [getTaxAdminData] peut lancer une exception ExceptionImpots
24      // on la laisse alors remonter au code appelant
25      $this->taxAdminData = $this->dao->getTaxAdminData();
26    }
27
28  // calcul de l'impôt
29  // ------------------------------------------------------------------------
30    public function calculerImpot(string $marié, int $enfants, int $salaire): array {
31      …
32      // résultat
33      return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" => $réduction,
    "taux" => $taux];
34    }
35
36  // ------------------------------------------------------------------------
37    private function calculerImpot2(string $marié, int $enfants, float $salaire): array {
38      …
39      // résultat
40      return ["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]];
41    }
42
43    // revenuImposable=salaireAnnuel-abattement
44    // l'abattement a un min et un max
45    private function getRevenuImposable(float $salaire): float {
46      …
47      // résultat
48      return floor($revenuImposable);
49    }
50
51  // calcule une décôte éventuelle
52    private function getDecôte(string $marié, float $salaire, float $impots): float {
53      …
54      // résultat
55      return ceil($décôte);
56    }
57
58  // calcule une réduction éventuelle
59    private function getRéduction(string $marié, float $salaire, int $enfants, float $impots): float {
60      ..
61      // résultat
62      return ceil($réduction);
63    }
64  }
```

Ce code a déjà été vu et commenté dès la version 1 au paragraphe lien. Sa version objet avec une base de données a été présentée au paragraphe lien.

1.18.2.4 Le script serveur





Le script serveur implémente la couche **[web] [4]**. Le script **[impots-server]** est configuré par le fichier jSON **[config-server.json]** suivant :

```
1   {
2       "rootDirectory": "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-08",
3       "databaseFilename": "Data/database.json",
4       "taxAdminDataFileName": "Data/taxadmindata.json",
5       "relativeDependencies": [
6           "/Entities/BaseEntity.php",
7           "/Entities/ExceptionImpots.php",
8           "/Entities/TaxAdminData.php",
9           "/Entities/Database.php",
10          "/Dao/InterfaceServerDao.php",
11          "/Dao/ServerDao.php",
12          "/Métier/InterfaceServerMetier.php",
13          "/Métier/ServerMetier.php"
14      ],
15      "absoluteDependencies": ["C:/myprograms/laragon-lite/www/vendor/autoload.php"],
16      "users": [
17          {
18              "login": "admin",
19              "passwd": "admin"
20          }
21      ]
22  }
```

- ligne 1 : le dossier racine à partir duquel les chemins de fichiers seront mesurés ;
- ligne 2 : le fichier jSON de configuration de la base de données MySQL ;
- ligne 3 : le fichier jSON des données de l'administration fiscale ;
- lignes 5-14 : les fichiers de l'application ;
- ligne 15 : la dépendance nécessaire aux bibliothèques tierces, ici Symfony ;
- lignes 16-20 : le tableau des utilisateurs autorisés à utiliser l'application ;

Les fichiers jSON **[database.json, taxadmindata.json]** sont ceux de la version 5 décrit au paragraphe lien.

Le script **[impots-server]** implémente la couche **[web]** de la façon suivante :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // gestion des erreurs par PHP
10  //ini_set("display_errors", "0");
11  //
12  // chemin du fichier de configuration
13  define("CONFIG_FILENAME", "Data/config-server.json");
14
15  // on récupère la configuration
16  $config = \json_decode(file_get_contents(CONFIG_FILENAME), true);
17
18  // on inclut les dépendances nécessaires au script
19  $rootDirectory = $config["rootDirectory"];
20  foreach ($config["relativeDependencies"] as $dependency) {
21    require "$rootDirectory$dependency";
22  }
```

```php
// dépendances absolues (bibliothèques tierces)
foreach ($config["absoluteDependencies"] as $dependency) {
  require "$dependency";
}

// définition des constantes
define("DATABASE_CONFIG_FILENAME", $config["databaseFilename"]);
//
// dépendances Symfony
use \Symfony\Component\HttpFoundation\Response;
use \Symfony\Component\HttpFoundation\Request;

// préparation de la réponse JSON du serveur
$response = new Response();
$response->headers->set("content-type", "application/json");
$response->setCharset("utf-8");

// on récupère la requête courante
$request = Request::createFromGlobals();
// authentification
$requestUser = $request->headers->get('php-auth-user');
$requestPassword = $request->headers->get('php-auth-pw');
// l'utilisateur existe-t-il ?
$users = $config["users"];
$i = 0;
$trouvé = FALSE;
while (!$trouvé && $i < count($users)) {
  $trouvé = ($requestUser === $users[$i]["login"] && $users[$i]["passwd"] === $requestPassword);
  $i++;
}
// on fixe le code de statut de la réponse
if (!$trouvé) {
  // pas trouvé - code 401
  $response->setStatusCode(Response::HTTP_UNAUTHORIZED);
  $response->headers->add(["WWW-Authenticate" => "Basic realm=" . utf8_decode("\"Serveur de calcul d'impôts\"")]);
  // msg d'erreur
  $response->setContent(\json_encode(["réponse" => ["erreur" => "Echec de l'authentification [$requestUser, $requestPassword]"]], JSON_UNESCAPED_UNICODE));
  $response->send();
  // fin
  exit;
}
// on a un utilisateur valide - on vérifie les paramètres reçus
$erreurs = [];
// on doit avoir trois paramètres GET
$method = strtolower($request->getMethod());
$erreur = $method !== "get" || $request->query->count() != 3;
// erreur ?
if ($erreur) {
  $erreurs[] = "Méthode GET requise avec les seuls paramètres [marié, enfants, salaire]";
}

// on récupère le statut marital
if (!$request->query->has("marié")) {
  $erreurs[] = "paramètre marié manquant";
} else {
  $marié = trim(strtolower($request->query->get("marié")));
  $erreur = $marié !== "oui" && $marié !== "non";
  // erreur ?
  if ($erreur) {
    $erreurs[] = "paramètre marié [$marié] invalide";
  }
}

// on récupère le nombre d'enfants
if (!$request->query->has("enfants")) {
  $erreurs[] = "paramètre enfants manquant";
} else {
  $enfants = trim($request->query->get("enfants"));
  // le nombre d'enfants doit être un nombre entier >=0
  $erreur = !preg_match("/^\d+$/", $enfants);
  // erreur ?
  if ($erreur) {
    $erreurs[] = "paramètre enfants [$enfants] invalide";
  }
}
```

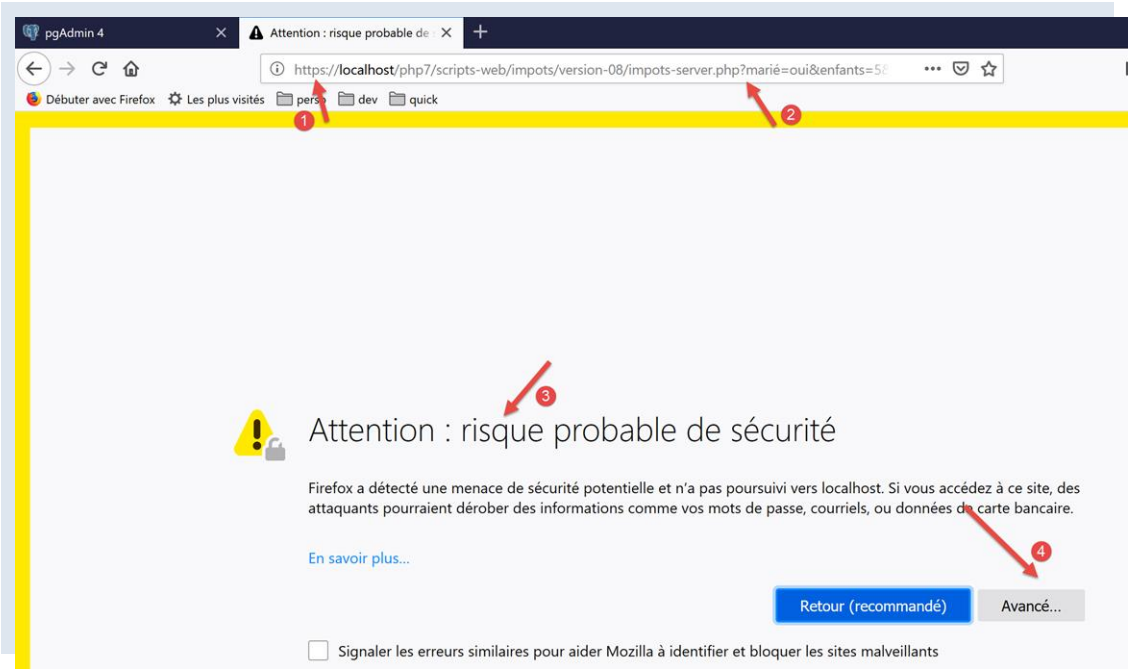
```
98
99  // on récupère le salaire annuel
100 if (!$request->query->has("salaire")) {
101   $erreurs[] = "paramètre salaire manquant";
102 } else {
103   // le salaire doit être un nombre entier >=0
104   $salaire = trim($request->query->get("salaire"));
105   $erreur = !preg_match("/^\d+$/", $salaire);
106   // erreur ?
107   if ($erreur) {
108     $erreurs[] = "paramètre salaire [$salaire] invalide";
109   }
110 }
111
112 // autres paramètres dans la requête ?
113 foreach (\array_keys($request->query->all()) as $key) {
114   // paramètre valide ?
115   if (!\in_array($key, ["marié", "enfants", "salaire"])) {
116     $erreurs[] = "paramètre [$key] invalide";}
117 }
118
119 // erreurs ?
120 if ($erreurs) {
121   // on envoie un code d'erreur 400 au client
122   $response->setStatusCode(Response::HTTP_BAD_REQUEST);
123   $response->setContent(json_encode(["réponse" => ["erreurs" => $erreurs]], JSON_UNESCAPED_UNICODE));
124   $response->send();
125   exit;
126 }
127 // on a tout ce qu'il faut pour travailler
128 // création de l'architecture du serveur
129 $msgErreur = "";
130 try {
131   // création de la couche [dao]
132   $dao = new ServerDao($config["databaseFilename"]);
133   // création de la couche [métier]
134   $métier = new ServerMetier($dao);
135 } catch (ExceptionImpots $ex) {
136 // on note l'erreur
137   $msgErreur = utf8_encode($ex->getMessage());
138 }
139 // erreur ?
140 if ($msgErreur) {
141   // on envoie un code d'erreur 500 au client
142   $response->setStatusCode(Response::HTTP_INTERNAL_SERVER_ERROR);
143   $response->setContent(\json_encode(["réponse" => ["erreur" => $msgErreur]], JSON_UNESCAPED_UNICODE));
144   $response->send();
145   exit;
146 }
147 // calcul de l'impôt
148 $result = $métier->calculerImpot($marié, (int) $enfants, (int) $salaire);
149 // on rend la réponse
150 $response->setContent(json_encode(["réponse" => $result], JSON_UNESCAPED_UNICODE));
151 $response->send();
```

Commentaires

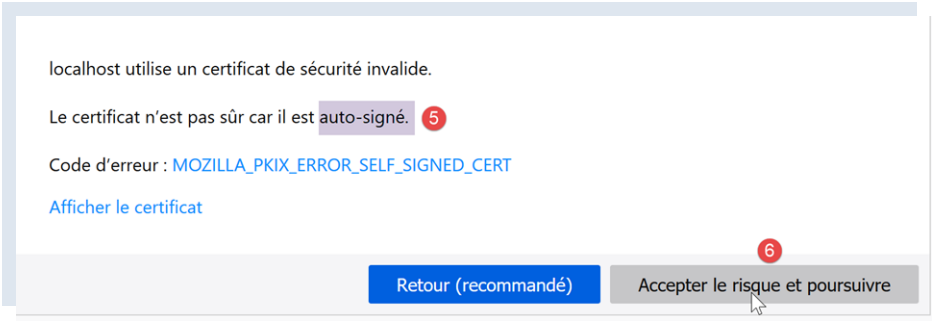
- ligne 16 : on exploite le fichier de configuration ;
- lignes 18-26 : on charge toutes les dépendances ;
- ligne 29 : le nom du fichier **[database.json]** ;
- lignes 32-33 : on déclare les classes des bibliothèques tierces qu'on va utiliser ;
- lignes 36-38 : on prépare une réponse jSON ;
- lignes 40-52 : on vérifie que l'utilisateur qui fait la requête fait bien partie des utilisateurs autorisés ;
- lignes 54-63 : si ce n'est pas le cas, on envoie le code HTTP 401 qui indique un refus d'accès. A réception de ce code et de l'entête HTTP **[WWW-Authenticate => Basic realm=]**, la plupart des navigateurs affichent une fenêtre d'authentification invitant l'utilisateur à s'authentifier ;
- ligne 59 : la réponse jSON du serveur explique la cause de l'erreur. Toutes les réponses du serveur seront la chaîne jSON d'un tableau **['réponse'=>'qq chose']** ;
- lignes 64-117 : on vérifie la validité de la requête :
  - une requête GET avec trois paramètres exactement ;
  - un paramètre **[marié]** dont la valeur doit être 'oui' ou 'non' ;
  - un paramètre **[enfants]** dont la valeur doit être un entier >=0 ;
  - un paramètre **[salaire]** dont la valeur doit être un entier >=0 ;

- ligne 65 : à chaque fois qu'une erreur est détectée, un message d'erreur est ajouté au tableau **[$erreurs]** ;
- lignes 120-126 : si erreur il y a, alors on envoie le code HTTP **[400 Bad Request]** au client (ligne 122) ;
- ligne 123 : la réponse jSON du serveur explique la cause de l'erreur ;
- à partir de la ligne 132, tout a été vérifié. On peut instancier les couches **[dao, métier]**. Cette instanciation a un coût et il ne faut la faire que si on est sûr d'avoir une requête valide ;
- lignes 130-138 : on crée l'architecture du serveur. La construction de la couche **[dao]** peut lancer une exception de type **[ExceptionImpots]**. Si cette exception se produit, on note l'erreur ;
- lignes 135-138 : si exception il y a eu, alors on envoie le code HTTP 500 au client. Ce code signifie que le serveur a bogué ;
- ligne 143 : la réponse explique la cause de l'erreur ;
- ligne 148 : le calcul de l'impôt est délégué à la couche **[métier]** ;
- lignes 150-151 : envoi de la réponse ;

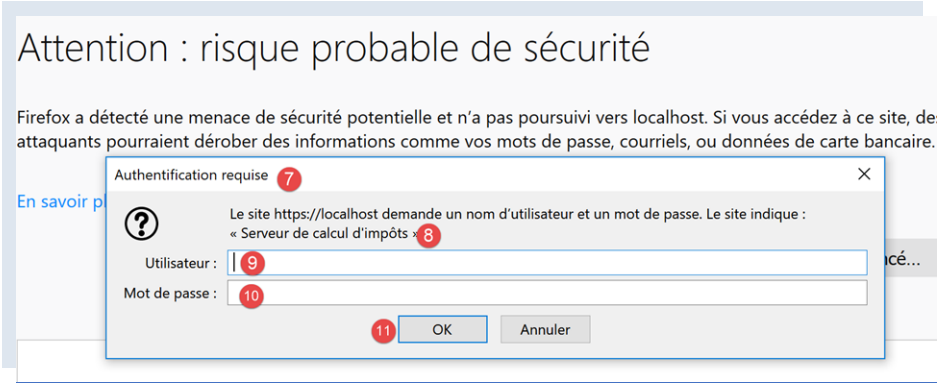
Testons ce script avec un navigateur. Demandons l'URL sécurisée **[https://localhost:443/php7/scripts-web/impots/version-08/impots-server.php?marié=oui&enfants=5&salaire=100000]**:



- en **[1]**, l'URL sécurisée demandée ;
- en **[2]**, les trois paramètres **[marié, enfants, salaire]** ;
- en **[3]**, le serveur Apache de Laragon a envoyé un certificat SSL autosigné. Le navigateur l'a remarqué et affiche un avertissement de sécurité : il considère que le site du serveur n'est pas digne de confiance ;
- en **[4]**, on continue ;



- en **[6]**, on continue ;



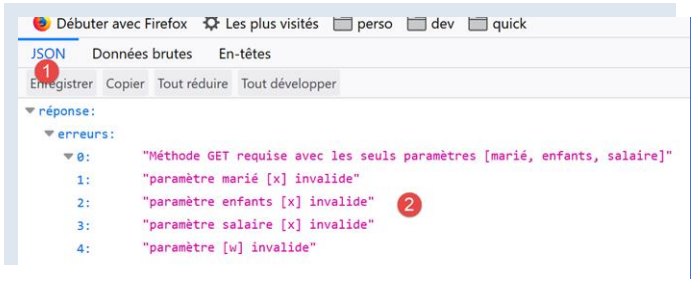
- en **[7]**, le navigateur affiche une fenêtre pour que l'utilisateur puisse s'authentifier ;
- en **[9,10]**, on tape **[admin]** et **[admin]** ;

- en **[13]**, la réponse jSON du serveur ;

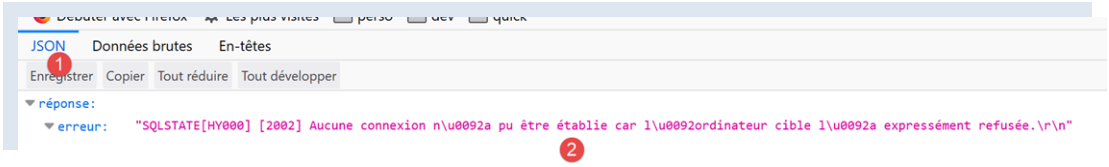
Faisons quelques tests d'erreur :

On demande l'URL **[https://localhost/php7/scripts-web/impots/version-08/impots-server.php?marié=x&enfants=x&salaire=x&w=x]**

On obtient le résultat suivant :



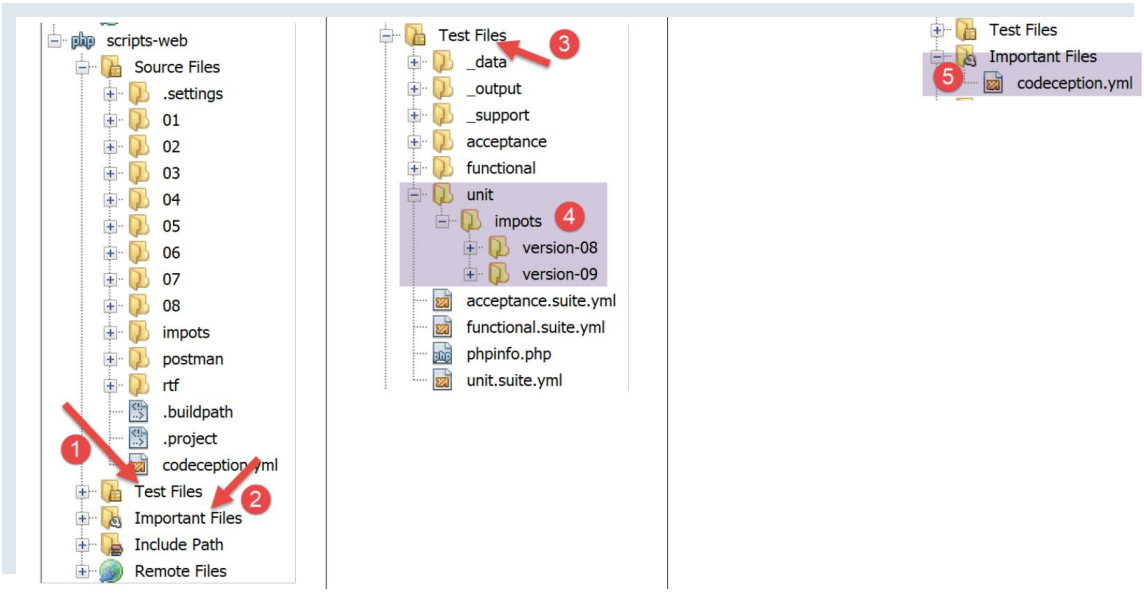
On coupe le SGBD MySQL et on demande l'URL **[https://localhost/php7/scripts-web/impots/version-08/impots-server.php?marié=oui&enfants=3&salaire=60000]** :



1.18.2.5 Tests [Codeception]

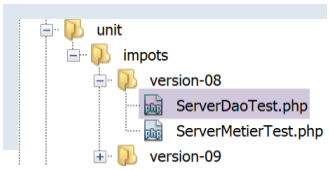
A chaque fois que nous construirons une nouvelle version du serveur, nous testerons les couches **[métier]** et **[dao]** comme il a été fait depuis la version 04 (cf paragraphes lien et lien).

Tout d'abord, nous associons le projet **[scripts-web]** aux tests **[Codeception]**. Pour cela, suivez la même procédure suivie pour le projet **[scripts-console]** au paragraphe lien. Nous obtenons un projet **[scripts-web]** avec un dossier **[Test Files]** :



Nous allons créer un test pour la couche **[dao]** et un pour la couche **[métier]**.

1.18.2.5.1 Tests de la couche [dao]



Le test **[ServerDaoTest]** sera le suivant :

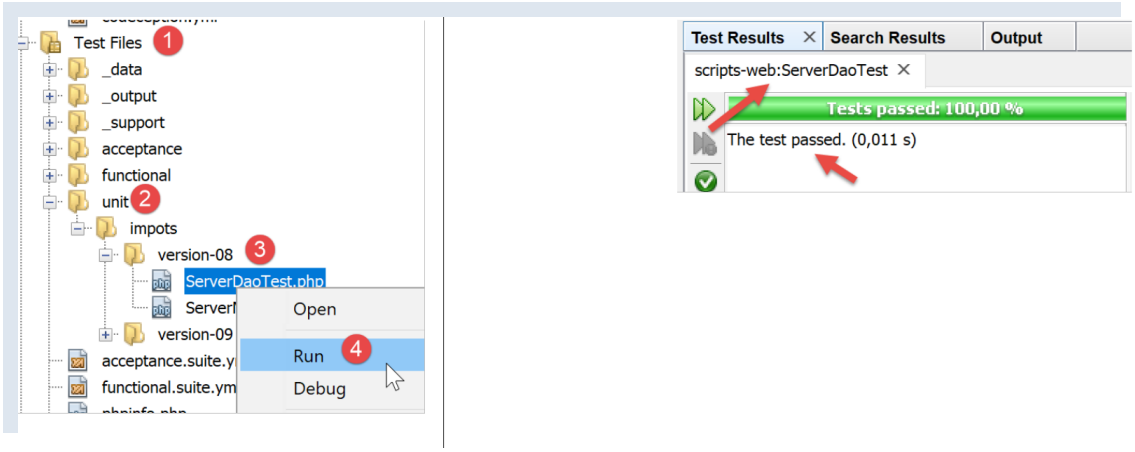
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // définition des constantes
10  define("ROOT", "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-08");
11  // chemin du fichier de configuration
12  define("CONFIG_FILENAME", ROOT . "/Data/config-server.json");
13
14  // on récupère la configuration
15  $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
16  // on inclut les dépendances nécessaires au script
17  $rootDirectory = $config["rootDirectory"];
18  foreach ($config["relativeDependencies"] as $dependency) {
19    require "$rootDirectory$dependency";
20  }
21  // dépendances absolues (bibliothèques tierces)
22  foreach ($config["absoluteDependencies"] as $dependency) {
23    require "$dependency";
24  }
25
26  // test ----------------------------------------------------
```

```
27
28  class ServerDaoTest extends \Codeception\Test\Unit {
29    // TaxAdminData
30    private $taxAdminData;
31
32    public function __construct() {
33      // parent
34      parent::__construct();
35      // on récupère la configuration
36      $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
37      // création de la couche [dao]
38      $dao = new ServerDao(ROOT . "/" . $config["databaseFilename"]);
39      $this->taxAdminData = $dao->getTaxAdminData();
40    }
41
42    // tests
43    public function testTaxAdminData() {
44      …
45    }
46
47  }
```

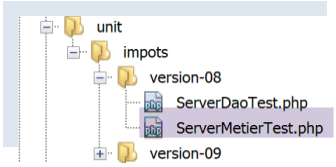
Commentaires

- lignes 9-24 : on construit le même environnement de travail que celui du serveur **[impots-server.php]**. Cela se fait en lignes 9-12 avec la définitions des deux constantes dont dépend l'environnement ;
- lignes 32-40 : on construit une instance de la couche **[dao]** à tester comme il était fait dans le script serveur **[impots-server.php]** ;
- à partir de maintenant on est dans les mêmes conditions que le script serveur **[impots-server.php]** : on peut démarrer les tests ;
- lignes 43-45 : la méthode **[testTaxAdminData]** est celle décrite au paragraphe lien ;

Les résultats du test sont les suivants :



1.18.2.5.2 Tests de la couche [métier]



Le test **[ServerMetierTest]** sera le suivant :

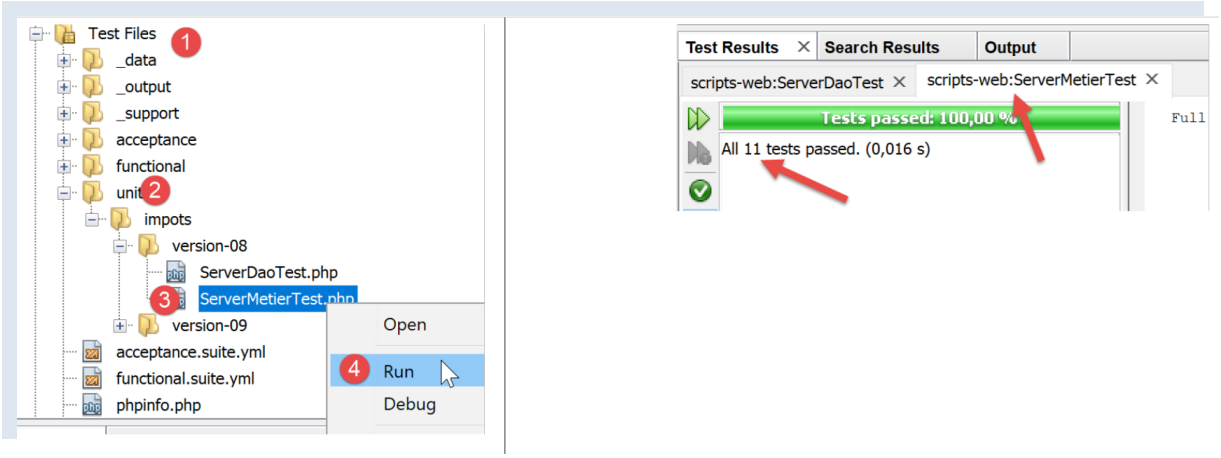
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
```

```
6   // espace de noms
7   namespace Application;
8
9   // définition des constantes
10  define("ROOT", "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-08");
11  // chemin du fichier de configuration
12  define("CONFIG_FILENAME", ROOT . "/Data/config-server.json");
13  // on récupère la configuration
14  $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
15  // on inclut les dépendances nécessaires au script
16  $rootDirectory = $config["rootDirectory"];
17  foreach ($config["relativeDependencies"] as $dependency) {
18    require "$rootDirectory$dependency";
19  }
20  // dépendances absolues (bibliothèques tierces)
21  foreach ($config["absoluteDependencies"] as $dependency) {
22    require "$dependency";
23  }
24
25  // classe de test
26  class ServerMetierTest extends \Codeception\Test\Unit {
27    // couche métier
28    private $métier;
29
30    public function __construct() {
31      parent::__construct();
32      // on récupère la configuration
33      $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
34      // création de la couche [dao]
35      $dao = new ServerDao(ROOT . "/" . $config["databaseFilename"]);
36      // création de la couche [métier]
37      $this->métier = new ServerMetier($dao);
38    }
39
40    // tests
41    public function test1() {
42     …
43    }
44
45    public function test2() {
46      …
47    }
48
49    ..
50
51    public function test11() {
52      …
53    }
54
55  }
```

Commentaires

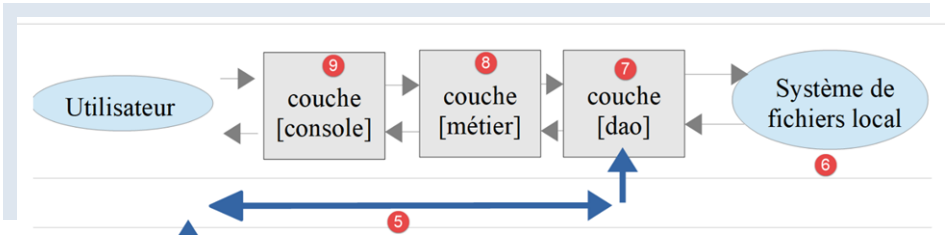
- lignes 9-24 : on construit le même environnement de travail que celui du serveur **[impots-server.php]**. Cela se fait en lignes 9-12 avec la définitions des deux constantes dont dépend l'environnement ;
- lignes 30-38 : on construit une instance de la couche **[métier]** à tester comme il était fait dans le script serveur **[impots-server.php]** ;
- à partir de maintenant on est dans les mêmes conditions que le script serveur **[impots-server.php]** : on peut démarrer les tests ;
- lignes 40-53 : les méthodes **[test1, test2…, test11]** sont celles décrites au paragraphe lien ;

Les résultats du test sont les suivants :

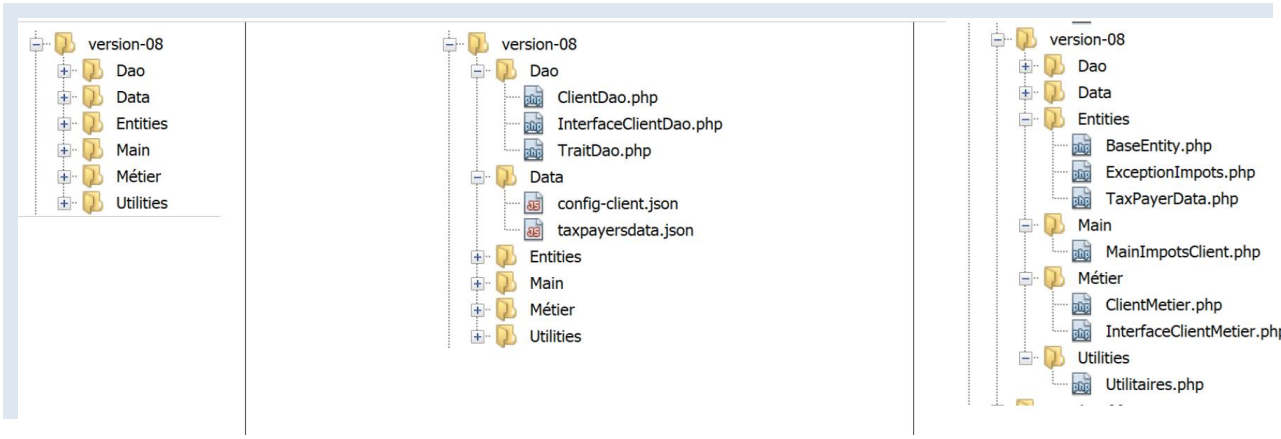


1.18.3 Le client

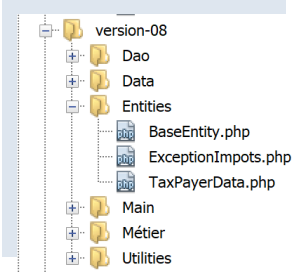
Nous nous intéressons à la partie cliente de l'application.



Cette architecture sera implémentée par les scripts suivants :



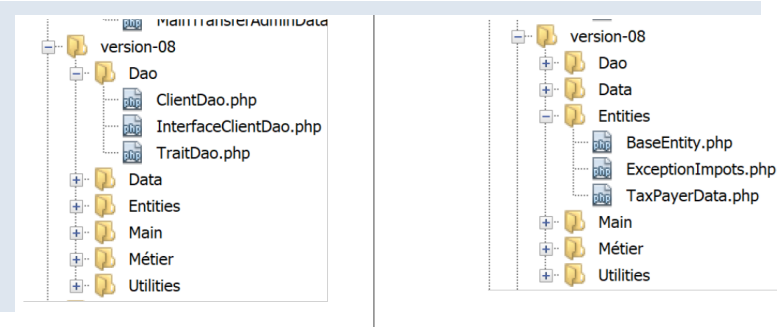
1.18.3.1 Les entités échangées entre couches



Les entités ci-dessus ont toutes été décrites et déjà utilisées :

- **[BaseEntity]** au paragraphe lien ;
- **[ExceptionImpots]** au paragraphe lien ;
- **[TaxPayerData]** au paragraphe lien ;

1.18.3.2 La couche [dao]



La couche **[dao]** implémente l'interface **[InterfaceClientDao]** suivante :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   interface InterfaceClientDao {
7
8     // lecture des données contribuables
9     public function getTaxPayersData(string $taxPayersFilename, string $errorsFilename): array;
10
11    // calcul des impôts d'un contribuable
12    public function calculerImpot(string $marié, int $enfants, int $salaire): array;
13
14    // enregistrement des résultats
15    public function saveResults(string $resultsFilename, array $taxPayersData): void;
16  }
```

- ligne 9 : la fonction **[getTaxPayersData]** amène en mémoire les données des contribuables du fichier **[$taxPayersFilename]**. Si erreurs il y a, elles sont consignées dans le fichier **[$errorsFilename]** ;
- ligne 12 : la fonction **[calculerImpots]** calcule l'impôt d'un contribuable ;
- ligne 15 : la fonction **[saveResults]** sauvegarde dans le fichier **[$resultsFilename]** les données du tableau **[$taxPayersData]** qui représentent les résultats de plusieurs calculs d'impôt ;

L'interface **[InterfaceClientDao]** est implémentée par la classe **[ClientDao]** suivante :

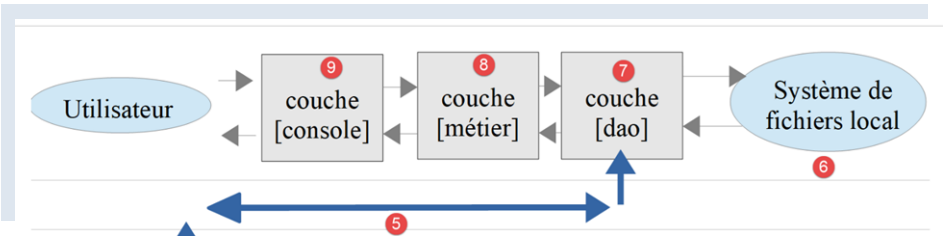
```
1   <?php
2
3   namespace Application;
4
5   // dépendances
6   use \Symfony\Component\HttpClient\HttpClient;
7
8   class ClientDao implements InterfaceClientDao {
9     // utilisation d'un Trait
10    use TraitDao;
11    // attributs
12    private $urlServer;
13    private $user;
14
15    // constructeur
16    public function __construct(string $urlServer, array $user) {
17      $this->urlServer = $urlServer;
18      $this->user = $user;
19    }
20
21    // calcul de l'impôt
22    public function calculerImpot(string $marié, int $enfants, int $salaire): array {
23      // on crée un client HTTP
24      $httpClient = HttpClient::create([
```

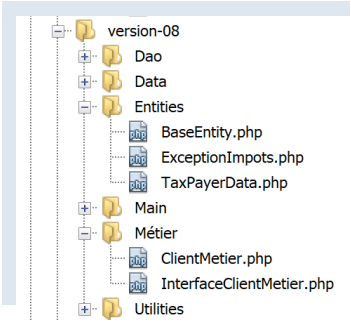
```
25          'auth_basic' => [$this->user["login"], $this->user["passwd"]],
26          "verify_peer" => false
27      ]);
28      // on fait la requête au serveur
29      $response = $httpClient->request('GET', $this->urlServer,
30        ["query" => [
31            "marié" => $marié,
32            "enfants" => $enfants,
33            "salaire" => $salaire
34      ]]);
35      // on récupère la réponse
36      $json = $response->getContent(false);
37      $array = \json_decode($json, true);
38      $réponse = $array["réponse"];
39      // logs
40      // print "$json=json\n";
41      // on récupère le statut de la réponse
42      $statusCode = $response->getStatusCode();
43      // erreur ?
44      if ($statusCode !== 200) {
45        // on a une erreur - on lance une exception
46        $réponse = ["statut HTTP" => $statusCode] + $réponse;
47        $message = \json_encode($réponse, JSON_UNESCAPED_UNICODE);
48        throw new ExceptionImpots($message);
49      }
50      // on rend la réponse
51      return $réponse;
52    }
53
54  }
```

Commentaires

- ligne 10 : on insère **[TraitDao]** (cf paragraphe lien) qui implémente les méthodes **[getTaxPayersData]** et **[saveResults]**. Ne reste donc que la méthode **[calculerImpots]** à implémenter. Celle-ci est implémentée aux lignes 22-49 ;
- lignes 16-19 : le constructeur de la classe **[ClientDao]** reçoit deux paramètres :
  - l'URL **[$urlServer]** du serveur de calcul d'impôt ;
  - le tableau **[$user]** de clés 'login' et 'passwd' qui définit l'utilisateur qui fait la requête ;
- ligne 22 : la méthode **[calculerImpots]** reçoit les trois paramètres à envoyer au serveur de calcul d'impôts ;
- lignes 24-27 : on crée un client HTTP avec :
  - ligne 25 : les identifiants de l'utilisateur qui fait la requête ;
  - ligne 26 : l'option qui fait que le client HTTP ne vérifiera pas la validité du certificat SSL envoyé par le serveur ;
- lignes 29-34 : le serveur est interrogé avec les trois paramètres qu'il attend ;
- ligne 36 : on récupère la réponse jSON du serveur. Si on ne met pas le paramètre **[false]** à la méthode **[Response::getContent]**, alors si le statut de la réponse du serveur est dans l'intervalle **[3xx-5xx]** (cas d'erreur), l'objet **[Response]** lance une exception dès qu'on cherche à obtenir le contenu de la réponse **[Response::getContent]** ou ses entêtes HTTP **[Response::getHeaders]**. Ici quelque soit le statut HTTP de la réponse, on veut pouvoir avoir accès au contenu de celle-ci, ne serait-ce que pour le loguer (ligne 40) ;
- lignes 37-38 : la réponse du serveur est la chaîne jSON d'un tableau **['réponse'=>qqChose]**. On récupère le **[qqChose]** ;
- ligne 40 : on logue la réponse jSON en mode développement ;
- ligne 42 : on récupère le code de statut de la réponse ;
- lignes 44-49 : si le code de statut HTTP n'est pas 200, alors c'est que notre serveur a rencontré un problème. On lance alors une exception de type **[ExceptionImpots]** avec pour message, la réponse jSON du serveur augmentée du code HTTP de la réponse ;
- ligne 51 : on rend le résultat qui est un tableau associatif avec les clés **[impôt, surcôte, décôte, réduction, taux]** ;

1.18.3.3 La couche [métier]





La couche **[métier] [8]** implémente l'interface **[InterfaceClientMetier]** suivante :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   interface InterfaceClientMetier {
7
8     // calcul des impôts d'un contribuable
9     public function calculerImpot(string $marié, int $enfants, int $salaire): array;
10
11    // calcul des impôts en mode batch
12    public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string
    $errorsFileName): void;
13  }
```

- ligne 9 : la fonction **[calculerImpots]** calcule l'impôt ;
- ligne 12 : la fonction **[executeBatchImpots]** calcule l'impôt des contribuables dont les données sont dans le fichier **[$taxPayersFileName]**, met les résultats obtenus dans le fichier **[$resultsFileName]** et les erreurs rencontrées dans le fichier **[$errorsFileName]** ;

L'interface **[InterfaceClientMetier]** est implémentée par la classe **[ClientMetier]** suivante :

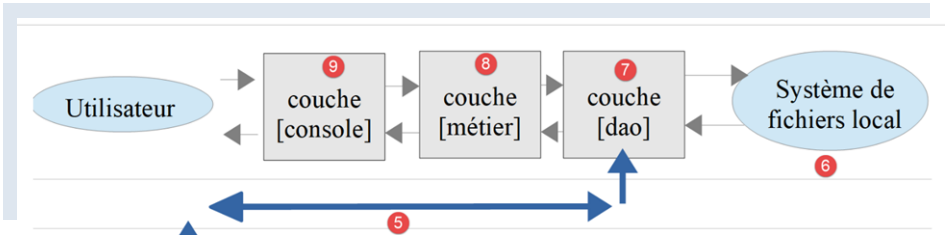
```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   class ClientMetier implements InterfaceClientMetier {
7     // attribut
8     private $clientDao;
9
10    // constructeur
11    public function __construct(InterfaceClientDao $clientDao) {
12      // on mémorise la référence sur la couche [dao]
13      $this->clientDao = $clientDao;
14    }
15
16    // calcul de l'impôt
17    public function calculerImpot(string $marié, int $enfants, int $salaire): array {
18      return $this->clientDao->calculerImpot($marié, $enfants, $salaire);
19    }
20
21    // calcul des impôts en mode batch
22    public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string
    $errorsFileName): void {
23      // on laisse remonter les exceptions qui proviennent de la couche [dao]
24      // on récupère les données contribuables
25      $taxPayersData = $this->clientDao->getTaxPayersData($taxPayersFileName, $errorsFileName);
26      // tableau des résultats
27      $results = [];
28      // on les exploite
29      foreach ($taxPayersData as $taxPayerData) {
30        // on calcule l'impôt
31        $result = $this->calculerImpot(
32          $taxPayerData->getMarié(),
33          $taxPayerData->getEnfants(),
34          $taxPayerData->getSalaire());
35        // on complète [$taxPayerData]
36        $taxPayerData->setFromArrayOfAttributes($result);
```

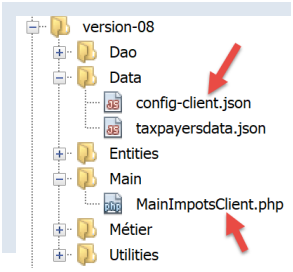
```
37          // on met le résultat dans le tableau des résultats
38          $results [] = $taxPayerData;
39      }
40      // enregistrement des résultats
41      $this->clientDao->saveResults($resultsFileName, $results);
42    }
43
44  }
```

Commentaires

- lignes 11-14 : le constructeur de la classe **[ClientMetier]** reçoit comme paramètre une référence sur la couche **[dao]** ;
- lignes 17-19 : le calcul de l'impôt est délégué à la couche **[dao]** ;
- lignes 20-38 : la fonction **[executeBatchImpots]** a été décrite au paragraphe lien ;

1.18.3.4 Le script principal





Le script client **[MainImpotsClient.php]** implémente la couche **[console]** [9]. Il est configuré par le fichier jSON **[conf-client.json]** suivant :

```
1   {
2       "rootDirectory": "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-08",
3       "taxPayersDataFileName": "Data/taxpayersdata.json",
4       "resultsFileName": "Data/results.json",
5       "errorsFileName": "Data/errors.json",
6       "dependencies": [
7           "Entities/BaseEntity.php",
8           "Entities/TaxPayerData.php",
9           "Entities/ExceptionImpots.php",
10          "Utilities/Utilitaires.php",
11          "Dao/InterfaceClientDao.php",
12          "Dao/TraitDao.php",
13          "Dao/ClientDao.php",
14          "Métier/InterfaceClientMetier.php",
15          "Métier/ClientMetier.php"
16      ],
17      "absoluteDependencies": [
18          "C:/myprograms/Laragon-Lite/www/vendor/autoload.php"
19      ],
20      "user": {
21          "login": "admin",
22          "passwd": "admin"
23      },
24      "urlServer": "https://localhost:443/php7/scripts-web/impots/version-08/impots-server.php"
25  }
```

- ligne 1 : le dossier racine du client ;
- ligne 2 : le fichier jSON des données contribuables ;
- ligne 3 : le fichier jSON des résultats ;

- ligen 4 : le fichier jSON des erreurs ;
- lignes 6-19 : les différentes dépendances du projet client ;
- lignes 20-23 : l'utilisateur faisant les requêtes au serveur de calcul d'impôts ;
- ligne 24 : l'URL sécurisée du serveur de calcul d'impôts ;

Le code du script **[MainImpotsClient.php]** est le suivant :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // gestion des erreurs par PHP
10  //ini_set("display_errors", "0");
11  //
12  // chemin du fichier de configuration
13  define("CONFIG_FILENAME", "../Data/config-client.json");
14
15  // on récupère la configuration
16  $config = \json_decode(file_get_contents(CONFIG_FILENAME), true);
17
18  // on inclut les dépendances nécessaires au script
19  $rootDirectory = $config["rootDirectory"];
20  foreach ($config["dependencies"] as $dependency) {
21    require "$rootDirectory/$dependency";
22  }
23  // dépendances absolues (bibliothèques tierces)
24  foreach ($config["absoluteDependencies"] as $dependency) {
25    require "$dependency";
26  }
27
28  // définition des constantes
29  define("TAXPAYERSDATA_FILENAME", "$rootDirectory/{$config["taxPayersDataFileName"]}");
30  define("RESULTS_FILENAME", "$rootDirectory/{$config["resultsFileName"]}");
31  define("ERRORS_FILENAME", "$rootDirectory/{$config["errorsFileName"]}");
32  //
33  // dépendances Symfony
34  use Symfony\Component\HttpClient\HttpClient;
35
36  // création de la couche [dao]
37  $clientDao = new ClientDao($config["urlServer"], $config["user"]);
38  // création de la couche [métier]
39  $clientMetier = new ClientMetier($clientDao);
40
41  // calcul de l'impôts en mode batch
42  try {
43    $clientMetier->executeBatchImpots(TAXPAYERSDATA_FILENAME, RESULTS_FILENAME, ERRORS_FILENAME);
44  } catch (\RuntimeException $ex) {
45    // on affiche l'erreur
46    print "L'erreur suivante s'est produite : " . $ex->getMessage() . "\n";
47  }
48  // fin
49  print "Terminé\n";
50  exit;
```

Commentaires

- ligne 13 : chemin du fichier de configuration ;
- ligne 16 : exploitation du fichier de configuration ;
- lignes 18-26 : chargement des dépendances ;
- ligne 37 : création de la couche **[dao]**. On passe au constructeur de la couche, les deux informations qu'il attend :
  - l'URL du serveur de calcul d'impôts ;
  - les identifiants de l'utilisateur qui va faire les requêtes ;
- ligne 39 : création de la couche **[métier]**. On passe au constructeur de la couche, une référence sur la couche **[dao]** qui vient d'être créée ;
- ligne 43 : on demande à la couche **[métier]** de :
  - calculer les impôts de tous les contribuables du fichier `$config["taxPayerDataFileName"]` ;
  - mettre les résultats dans le fichier `$config["resultsFileName"]` ;
  - mettre les erreurs dans le fichier `$config["errorsFileName"]` ;
- la ligne 43 peut lancer des exceptions ;

- ligne 46 : affichage du message d'erreur de l'exception ;

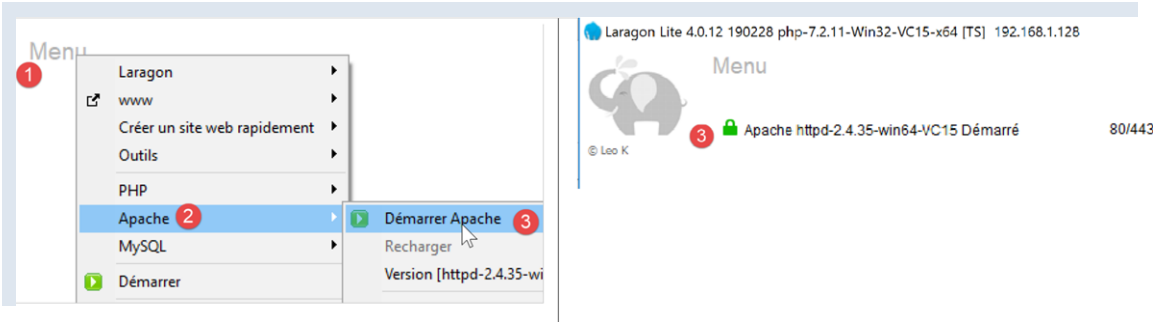
L'exécution du client amène les mêmes résultats que les versions précédentes. Vérifiez les fichiers suivants :

- **[Data/taxpayersdata.json]** : données des contribuables pour lesquel on calcule le montant de l'impôt ;
- **[Data/results.json]** : résultats pour les différents contribuables du fichier **[Data/taxpayersdata.json]** ;
- **[Data/errors.json]** : les erreurs qui ont pu être rencontrées dans l'exploitation du fichier **[Data/taxpayersdata.json]** ;

Regardons les cas d'erreur possibles. Tout d'abord, arrêtons le serveur Laragon. Les résultats dans la console du client sont alors les suivants :

```
1.  Couldn't connect to server for"https://localhost/php7/scripts-web/impots/version-08/impots-
    server.php?mari%C3%A9=oui&enfants=2&salaire=55555".
2.  Terminé
```

Maintenant lançons seulement le serveur Apache et pas le SGBD MySQL :



Les résultats dans la console du client sont alors les suivants :

```
1.  L'erreur suivante s'est produite : {"statut HTTP":500,"erreur":"SQLSTATE[HY000] [2002] Aucune connexion n'a
    pu être établie car l'ordinateur cible l'a expressément refusée.\r\n"}
2.  Terminé
```

Maintenant, lançons MySQL puis modifions dans **[config-client]** l'utilisateur qui se connecte :

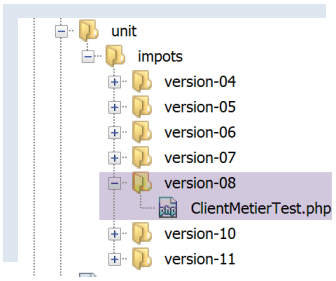
```
1       "user": {
2           "login": "x",
3           "passwd": "x"
4   },
```

Les résultats dans la console du client sont alors les suivants :

```
1.  L'erreur suivante s'est produite : {"statut HTTP":401,"erreur":"Echec de l'authentification [x, x]"}
2.  Terminé
```

1.18.3.5 Tests [Codeception]

Comme il a été fait pour les version précédentes, nous allons écrire des tests **[Codeception]** pour la version 08.



1.18.3.5.1 Test de la couche [métier]

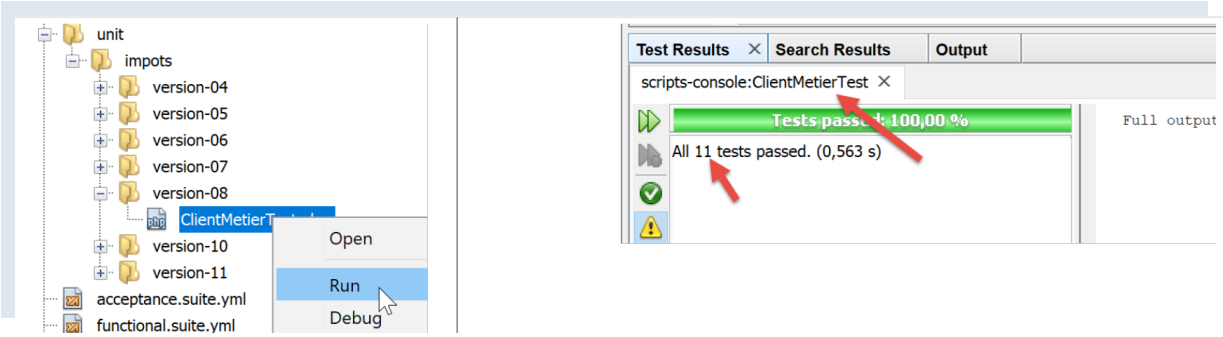
Le test **[ClientMetierTest.php]** est le suivant :

```php
<?php

// respect strict des types déclarés des paramètres de foctions
declare (strict_types=1);

// espace de noms
namespace Application;

// définition des constantes
define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-08");

// chemin du fichier de configuration
define("CONFIG_FILENAME", ROOT . "/Data/config-client.json");

// on récupère la configuration
$config = \json_decode(file_get_contents(CONFIG_FILENAME), true);

// on inclut les dépendances nécessaires au script
$rootDirectory = $config["rootDirectory"];
foreach ($config["dependencies"] as $dependency) {
  require "$rootDirectory/$dependency";
}
// dépendances absolues (bibliothèques tierces)
foreach ($config["absoluteDependencies"] as $dependency) {
  require "$dependency";
}
//
// classe de test
class ClientMetierTest extends \Codeception\Test\Unit {
  // couche métier
  private $métier;

  public function __construct() {
    parent::__construct();
    // on récupère la configuration
    $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
    // création de la couche [dao]
    $clientDao = new ClientDao($config["urlServer"], $config["user"]);
    // création de la couche [métier]
    $this->métier = new ClientMetier($clientDao);
  }

  // tests
  public function test1() {
    …
  }

  -------------

  public function test11() {
    …
  }

}
```

Commentaires

- lignes 10-26 : définition de l'environnement du test. Nous utilisons le même que celui utilisé par le script principal **[MainImpotsClient]** décrit au paragraphe lien ;
- lignes 33-41 : construction des couches **[dao]** et **[métier]** ;
- ligne 40 : l'attribut **[$this→métier]** référence la couche **[métier]** ;
- lignes 44-51 : les méthodes **[test1, test2…, test11]** sont celles décrites au paragraphe lien ;

Les résultats du test sont les suivants :



1.19 Exercice d'application – version 9

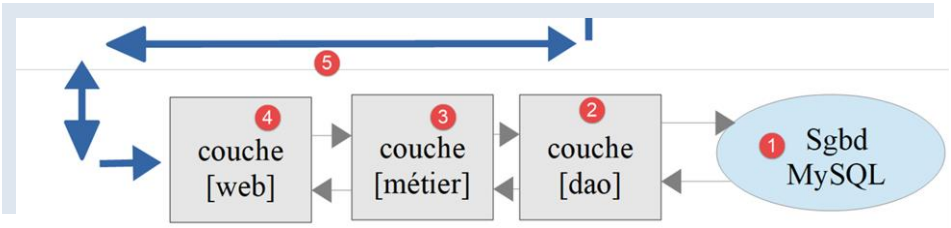
Dans cette version, nous allons améliorer le serveur de la façon suivante :

- actuellement, à chaque requête les données de l'administration fiscale sont cherchées dans la base de données. Nous allons utiliser une session :
  - lors de la 1re requête d'un utilisateur, de l'administration fiscale sont cherchées dans la base de données et mises en session ;
  - lors des requêtes suivantes du même utilisateur, de l'administration fiscale sont cherchées dans la session. On peut espérer un léger gain de temps d'exécution car les requêtes en base de données coûtent cher ;
- le serveur va loguer dans un fichier texte, les moments importants :
  - l'authentification réussie ou ratée ;
  - la validité ou non des paramètres envoyés par le client ;
  - le résultat du calcul d'impôt ;
  - les différents cas d'erreur ;
- en cas d'erreur fatale, un mail sera envoyé à l'administrateur de l'application ;

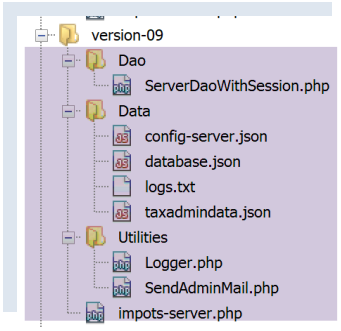
Le client devra lui aussi être modifié pour gérer le cookie de session qu'on va lui envoyer.

1.19.1 Le serveur

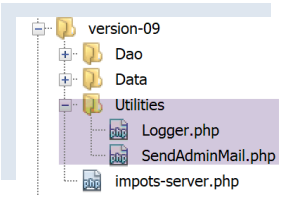
Nous nous intéressons à la partie serveur de l'application.



Cette architecture sera implémentée par les scripts suivants :



1.19.1.1 Utilitaires



1.19.1.1.1 La classe `[Logger]`

La classe **`[Logger]`** sera utilisée pour écrire des logs dans un fichier texte :

```
1   <?php
2
3   namespace Application;
4
5   class Logger {
6     // attribut
7     private $resource;
8
9     // constructeur
10    public function __construct(string $logsFilename) {
11      // ouverture du fichier
12      $this->resource = fopen($logsFilename, "a");
13      if (!$this->resource) {
14        throw new ExceptionImpots("Echec lors de la création du fichier de logs [$logsFilename]");
15      }
16    }
17
18    // écriture d'un message dans les logs
19    public function write(string $message) {
20      fputs($this->resource, (new \DateTime())->format("d/m/y H:i:s:v") . " : $message");
21    }
22
23    // fermeture du fichier des logs
24    public function close() {
25      fclose($this->resource);
26    }
27
28  }
```

Commentaires

- ligne 7 : la ressource du fichier de logs ;
- ligne 10 : le constructeur de la classe reçoit comme paramètre le nom du fichier de logs ;
- ligne 12 : on ouvre le fichier texte en mode ajout (a+) : le fichier sera ouvert, son contenu préservé. Les écritures se feront derrière le contenu actuel ;
- lignes 13-15 : si l'ouverture n'a pu se faire, une exception est lancée ;
- lignes 19-21 : la méthode **`[write]`** permet d'écrire le message **`[$message]`** dans le fichier de logs, précédé de la date et de l'heure ;
- lignes 24-16 : la méthode **`[close]`** permet permet de fermer le fichier de logs ;

Note : l'application serveur est susceptible de servir plusieurs clients simultanément. Or on a un seul fichier de logs pour tous. Il y a donc un risque d'accès concurrents pour écrire dans le fichier. Il faudrait donc synchroniser les écritures pour éviter que celles-ci ne se mélangent. Pour cela PHP dispose de sémaphores **`[https://www.php.net/manual/fr/book.sem.php]`**. Nous ignorerons la synchronisation des écritures ici mais il faut rester conscient du problème.

1.19.1.1.2 La classe `[SendAdminMail]`

La classe **`[SendAdminMail]`** permet d'envoyer un mail à l'administrateur de l'application en cas de plantage de celle-ci :

```
1   <?php
2
3   namespace Application;
4
5   class SendAdminMail {
6     // attributs
7     private $config;
8     private $logger;
9
10    // constructeur
```

```php
11    public function __construct(array $config, Logger $logger = NULL) {
12      $this->config = $config;
13      $this->logger = $logger;
14    }
15
16    public function send() {
17      // envoie $this->config['message'] au serveur smtp $this->config['smtp-server'] sur le port $infos[smt-port]
18      // si $this->config['tls'] est vrai, le support TLS sera utilisé
19      // le mail est envoyé de la part de $this->config['from']
20      // pour le destinataire $this->config['to']
21      // le message a le sujet $this->config['subject']
22      // on attache au mail les attachements de $this->config['attachments']
23      // le résultat de la méthode
24      try {
25        // création du message
26        $message = (new \Swift_Message())
27          // sujet du message
28          ->setSubject($this->config["subject"])
29          // expéditeur
30          ->setFrom($this->config["from"])
31          // destinataires avec un dictionnaire (setTo/setCc/setBcc)
32          ->setTo($this->config["to"])
33          // texte du message
34          ->setBody($this->config["message"])
35        ;
36        // attachements
37        foreach ($this->config["attachments"] as $attachment) {
38          // chemin de l'attachement
39          $fileName = __DIR__ . $attachment;
40          // on vérifie que le fichier existe
41          if (file_exists($fileName)) {
42            // on attache le document au message
43            $message->attach(\Swift_Attachment::fromPath($fileName));
44          } else {
45            if ($this->logger !== NULL) {
46              // erreur
47              $this->logger->write("L'attachement [$fileName] n'existe pas\n");
48            }
49          }
50        }
51        // protocole TLS ?
52        if ($this->config["tls"] === "TRUE") {
53          // TLS
54          $transport = (new \Swift_SmtpTransport($this->config["smtp-server"], $this->config["smtp-port"], 'tls'))
55            ->setUsername($this->config["user"])
56            ->setPassword($this->config["password"]);
57        } else {
58          // pas de TLS
59          $transport = (new \Swift_SmtpTransport($this->config["smtp-server"], $this->config["smtp-port"]));
60        }
61        // le gestionnaire de l'envoi
62        $mailer = new \Swift_Mailer($transport);
63        // envoi du message
64        $mailer->send($message);
65        // fin
66        if ($this->logger !== NULL) {
67          $this->logger->write("Message [{$this->config["message"]}] envoyé à {$this->config["to"]}\n");
68        }
69      } catch (\Throwable $ex) {
70        // erreur
71        if ($this->logger !== NULL) {
72          $this->logger->write("Erreur lors de l'envoi du message [{$this->config["message"]}] à
    {$this->config["to"]}\n");
73        }
74      }
75    }
76
77  }
```

Commentaires

- ligne 11 : le constructeur reçoit deux paramètres :
  - **[$config]** : un tableau associatif contenant toutes les informations nécessaires à l'envoi du mail ;
  - **[$logger]** : un logger permettant de loguer les moments importants de l'envoi du mail ;

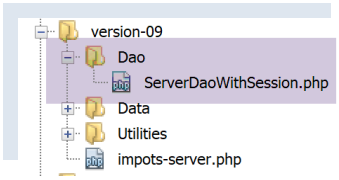
Le tableau associatif aura la forme suivante :

```
1.  {
```

```
2.    "smtp-server": "localhost",
3.    "smtp-port": "25",
4.    "from": "guest@localhost",
5.    "to": "guest@localhost",
6.    "subject": "plantage du serveur de calcul d'impôts",
7.    "tls": "FALSE",
8.    "attachments": []
9.  }
```

- lignes 16-76 : la méthode **[send]** permet d'envoyer le mail. Ce code a été présenté et décrit au paragraphe lien ;

1.19.1.2 La couche [dao]



Le script **[ServeurDaoWithSession.php]** est le suivant :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   // définition d'une classe ImpotsWithDataInDatabase
7   class ServerDaoWithSession extends ServerDao {
8
9     // constructeur
10    public function __construct(string $databaseFilename = NULL, TaxAdminData $taxAdminData = NULL) {
11      // cas le + simple
12      if ($taxAdminData !== NULL) {
13        $this->taxAdminData = $taxAdminData;
14      } else {
15        // on passe la main à la classe parent
16        parent::__construct($databaseFilename);
17      }
18    }
19
20  }
```

Commentaires

- ligne 7 : la classe **[ServerDaoWithSession]** de la version 09 étend la classe **[ServerDao]** de la version 08. En effet, la classe **[ServerDao]** sait utiliser la base de données. Il ne nous reste plus qu'à prévoir le cas où les données de l'administration fiscale ont déjà été acquises :
- ligne 10 : le constructeur reçoit maintenant deux paramètres :
  - **[string $databaseFilename]** : nom du fichier contenant les informations permettant de se connecter à la base de données si les données de l'administration fiscale n'ont pas encore été acquises, NULL sinon ;
  - **[TaxAdminData $taxAdminData]** : les données de l'administration fiscale si déjà acquises, NULL sinon ;

Lors du démarrage d'une session web, la couche **[dao]** sera construite avec un objet **[$databaseFilename]** non NULL et un objet **[taxAdminData]** NULL. Les données de l'administration fiscale seront alors recherchées en base et mémorisées dans la session. Lors des requêtes ultérieures de la même session, la couche **[dao]** sera construite avec un objet **[databaseFilename]** NULL et un objet **[taxAdminData]** provenant de la session et non NULL. Il n'y aura donc pas de recherche en base.

1.19.1.3 Le script serveur

Le script serveur **[impots-server.php]** est configuré par le fichier jSON **[config-server.json]** suivant :

```
1   {
2       "rootDirectory": "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-09",
3       "databaseFilename": "Data/database.json",
4       "relativeDependencies": [
5           "/../version-08/Entities/BaseEntity.php",
6           "/../version-08/Entities/ExceptionImpots.php",
7           "/../version-08/Entities/TaxAdminData.php",
```

```
            "/../version-08/Entities/Database.php",
            "/../version-08/Dao/InterfaceServerDao.php",
            "/../version-08/Dao/ServerDao.php",
            "/Dao/ServerDaoWithSession.php",
            "/../version-08/Métier/InterfaceServerMetier.php",
            "/../version-08/Métier/ServerMetier.php",
            "/Utilities/Logger.php",
            "/Utilities/SendAdminMail.php"
        ],
        "absoluteDependencies": ["C:/myprograms/laragon-lite/www/vendor/autoload.php"],
        "users": [
            {
                "login": "admin",
                "passwd": "admin"
            }
        ],
        "adminMail": {
            "smtp-server": "localhost",
            "smtp-port": "25",
            "from": "guest@localhost",
            "to": "guest@localhost",
            "subject": "plantage du serveur de calcul d'impôts",
            "tls": "FALSE",
            "attachments": []
        },
        "logsFilename": "Data/logs.txt"
    }
```

Le script serveur **[impots-server.php]** évolue de la façon suivante :

```
    <?php

    // respect strict des types déclarés des paramètres de foctions
    declare (strict_types=1);

    // espace de noms
    namespace Application;

    // gestion des erreurs par PHP
    ini_set("display_errors", "0");
    //
    // chemin du fichier de configuration
    define("CONFIG_FILENAME", "Data/config-server.json");

    // on récupère la configuration
    $config = \json_decode(file_get_contents(CONFIG_FILENAME), true);

    // on inclut les dépendances nécessaires au script
    $rootDirectory = $config["rootDirectory"];
    foreach ($config["relativeDependencies"] as $dependency) {
      require "$rootDirectory$dependency";
    }
    // dépendances absolues (bibliothèques tierces)
    foreach ($config["absoluteDependencies"] as $dependency) {
      require "$dependency";
    }
    //
    // dépendances Symfony
    use \Symfony\Component\HttpFoundation\Response;
    use \Symfony\Component\HttpFoundation\Request;
    use \Symfony\Component\HttpFoundation\Session\Session;

    // session
    $session = new Session();
    $session->start();

    // préparation de la réponse JSON du serveur
    $response = new Response();
    $response->headers->set("content-type", "application/json");
    $response->setCharset("utf-8");

    // création du fichier des logs
    try {
      $logger = new Logger($config['logsFilename']);
    } catch (ExceptionImpots $ex) {
      // internal server error
      doInternalServerError($ex->getMessage(), $response, NULL, $config['adminMail']);
```

```
48    // terminé
49    exit;
50  }
51
52  // 1er log
53  $logger->write("\n---nouvelle requête\n");
54
55  // on récupère la requête courante
56  $request = Request::createFromGlobals();
57  // authentification seulement la 1re fois
58  if (!$session->has("user")) {
59    // log
60    $logger->write("Autentification en cours…\n");
61    // authentification
62    …
63    }
64    // a-t-on trouvé l'utilisateur ?
65    if (!$trouvé) {
66      // pas trouvé - code 401 HTTP_UNAUTHORIZED
67      sendResponse(
68        $response,
69        ["erreur" => "Echec de l'authentification [$requestUser, $requestPassword]"],
70        Response::HTTP_UNAUTHORIZED,
71        ["WWW-Authenticate" => "Basic realm=" . utf8_decode("\"Serveur de calcul d'impôts\"")],
72        $logger
73      );
74      // terminé
75      exit;
76    } else {
77      // on note dans la session qu'on a authentifié l'utilisateur
78      $session->set("user", TRUE);
79      // log
80      $logger->write("Authentification réussie [$requestUser, $requestPassword]\n");
81    }
82  } else {
83    // log
84    $logger->write("Authentification prise en session…\n");
85  }
86  // on a un utilisateur valide - on vérifie les paramètres reçus
87  $erreurs = [];
88  // on doit avoir trois paramètres GET
89  …
90
91  // erreurs ?
92  if ($erreurs) {
93  // on envoie un code d'erreur 400 HTTP_BAD_REQUEST au client
94    sendResponse($response, ["erreurs" => $erreurs], Response::HTTP_BAD_REQUEST, [], $logger);
95    // terminé
96    exit;
97  } else {
98    // logs
99    $logger->write("paramètres ['marié'=>$marié, 'enfants'=>$enfants, 'salaire'=>$salaire] valides\n");
100 }
101 // on a tout ce qu'il faut pour travailler
102 // création de la couche [dao]
103 if (!$session->has("taxAdminData")) {
104   // les données sont prises dans la base de données
105   $logger->write("données fiscales prises en base de données\n");
106   try {
107     // construction de la couche [dao]
108     $dao = new ServerDaoWithSession($config["databaseFilename"], NULL);
109     // on met les données en session
110     $session->set("taxAdminData", $dao->getTaxAdminData());
111   } catch (\RuntimeException $ex) {
112     // on note l'erreur
113     doInternalServerError(utf8_encode($ex->getMessage()), $response, $logger, $config['adminMail']);
114     // terminé
115     exit;
116   }
117 } else {
118   // les données sont prises dans la session
119   $dao = new ServerDaoWithSession(NULL, $session->get("taxAdminData"));
120   // logs
121   $logger->write("données fiscales prises en session\n");
122 }
123 // création de la couche [métier]
124 $métier = new ServerMetier($dao);
```

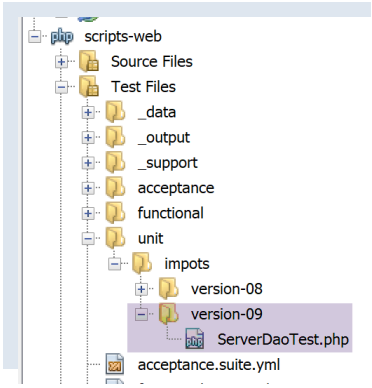
```
125 // calcul de l'impôt
126 $result = $métier->calculerImpot($marié, (int) $enfants, (int) $salaire);
127 // on rend la réponse
128 sendResponse($response, $result, Response::HTTP_OK, [], $logger);
129 // fin
130 exit;
131
132 function doInternalServerError(string $message, Response $response, Logger $logger = NULL, array $infos) {
133   // on envoie un mail à l'administrateur
134   // SendAdminMail intercepte toutes les exception et les logue lui-même
135   $infos['message'] = $message;
136   $sendAdminMail = new SendAdminMail($infos, $logger);
137   $sendAdminMail->send();
138   // on envoie un code d'erreur 500 au client
139   sendResponse($response, ["erreur" => $message], Response::HTTP_INTERNAL_SERVER_ERROR, [], $logger);
140 }
141
142 // fonction d'envoi de la réponse HTTP au client
143 function sendResponse(Response $response, array $result, int $statusCode, array $headers, Logger $logger) {
144   // $response : réponse HTTP
145   // $result : tableau des résultats
146   // $statusCode : statut HTTP de la réponse
147   // $headers : entêtes HTTP à mettre dans la réponse
148   // $logger : le logueur de l'application
149   //
150   // statut HTTTP
151   $response->setStatusCode($statusCode);
152   // body
153   $body = \json_encode(["réponse" => $result], JSON_UNESCAPED_UNICODE);
154   $response->setContent($body);
155   // headers
156   $response->headers->add($headers);
157   // envoi
158   $response->send();
159   // log
160   if ($logger != NULL) {
161     $logger->write("$body\n");
162     $logger->close();
163   }
164 }
```

Commentaires

- lignes 34-35 : on démarre une session ;
- lignes 38-40 : on prépare une réponse jSON ;
- lignes 42-50 : on essaie de créer le fichier de logs. Si exception il y a, la méthode **[doInternalServer]** (lignes 132-140) est appelée ;
- ligne 132 : la méthode **[doInternalServer]** accepte quatre paramètres :
  - **[$message]** : le message à loguer. Doit être codé en UTF-8 ;
  - **[$response]** : l'objet **[Response]** qui encapsule la réponse du serveur à son client ;
  - **[$logger]** : l'objet **[Logger]** permettant de faire les logs ;
  - **[$infos]** : les informations permettant d'envoyer un mail à l'administrateur de l'application ;
- lignes 135-137 : on envoie un mail à l'administrateur de l'application ;
- ligne 139 : on envoie la réponse au client :
  - $response : réponse HTTP ;
  - $result : le serveur envoie la chaîne jSON du tableau **['réponse'=>["erreur" => $message]]** ;
  - $statusCode : **[Response::HTTP_INTERNAL_SERVER_ERROR]**, code 500 ;
  - $headers : **[]**, pas d'entêtes HTTP à ajouter à la réponse ;
  - $logger : le logueur de l'application ;
- ligne 58 : grâce à la session mise en place on ne fera l'authentification du client qu'une seule fois :
  - une fois le client authentifié, on mettra une clé **[user]** dans la session (ligne 78) ;
  - lors de la requête suivante du même client, la ligne 58 évite une authentification devenue inutile ;
- ligne 103 : grâce à la session mise en place on ne cherchera les données en base qu'une seule fois :
  - lors de la première requête, la recherche en base se fera (ligne 108). Les données récupérées sont ensuite mises en session (ligne 110) associées à la clé **[taxAdminData]** ;
  - lors des requêtes suivantes, la clé **[taxAdminData]** sera trouvée en session (ligne 103) et alors les données discales seront directement communiquées à la couche **[dao]** (ligne 119) ;
- lignes 111-116 : la recherche des données fiscales en base peut échouer. Dans ce cas, on envoie au client un code **[500 Internal Server Error]** ;
- ligne 113 : le message d'erreur de l'exception du pilote MySQL est codé en ISO 8859-1. On le convertit en UTF-8 pour être correctement logué ;

- le reste du code est quasi identique à celui de la version précédente ;
- lignes 143-164 : la fonction **[sendResponse]** envoie toutes les réponses au client ;
- lignes 144-148 : signification des paramètres ;
- ligne 153 : la réponse est toujours la chaîne jSON d'un tableau **['résultat'=>qqChose]** ;
- ligne 156 : parfois il y a des entêtes HTTP à ajouter à la réponse. C'est le cas en ligne 71 ;
- ligne 158 : la réponse est envoyée ;
- lignes 160-163 : la réponse est loguée et le logueur fermé ;

1.19.1.4 Tests [Codeception]



Nous n'allons tester que la couche **[dao]** qui est la seule à avoir changé.

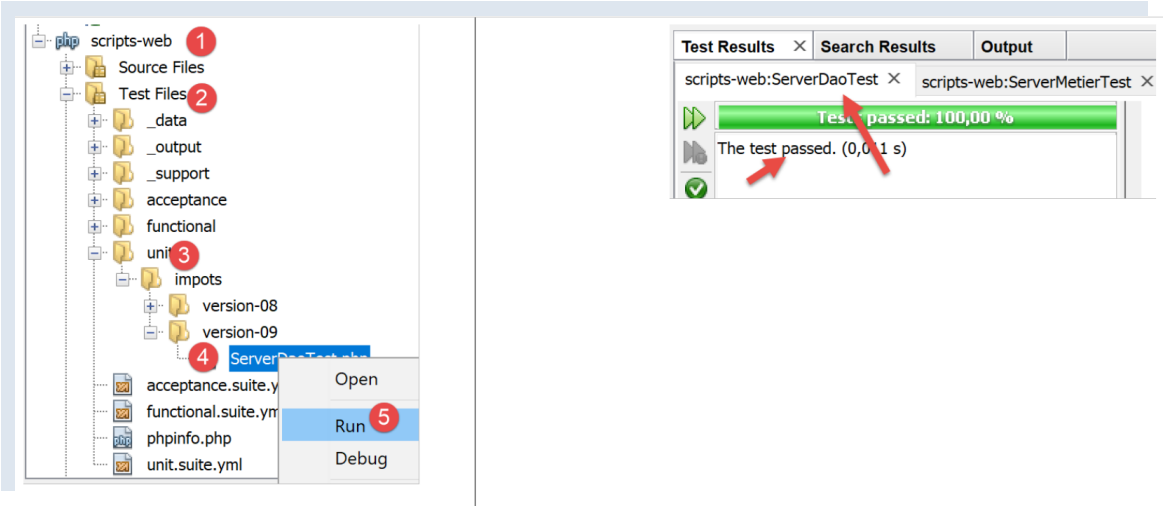
Le code du test **[ServerDaoTest]** est le suivant :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // définition des constantes
10  define("ROOT", "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-09");
11  // chemin du fichier de configuration
12  define("CONFIG_FILENAME", ROOT . "/Data/config-server.json");
13
14  // on récupère la configuration
15  $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
16  // on inclut les dépendances nécessaires au script
17  $rootDirectory = $config["rootDirectory"];
18  foreach ($config["relativeDependencies"] as $dependency) {
19    require "$rootDirectory$dependency";
20  }
21  // dépendances absolues (bibliothèques tierces)
22  foreach ($config["absoluteDependencies"] as $dependency) {
23    require "$dependency";
24  }
25
26  // test ------------------------------------------------------
27
28  class ServerDaoTest extends \Codeception\Test\Unit {
29    // TaxAdminData
30    private $taxAdminData;
31
32    public function __construct() {
33      // parent
34      parent::__construct();
35      // on récupère la configuration
36      $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
37      // création de la couche [dao]
38      $dao = new ServerDaoWithSession(ROOT . "/" . $config["databaseFilename"]);
39      $this->taxAdminData = $dao->getTaxAdminData();
40    }
41
42    // tests
```

```
43      public function testTaxAdminData() {
44          …
45      }
46
47  }
```

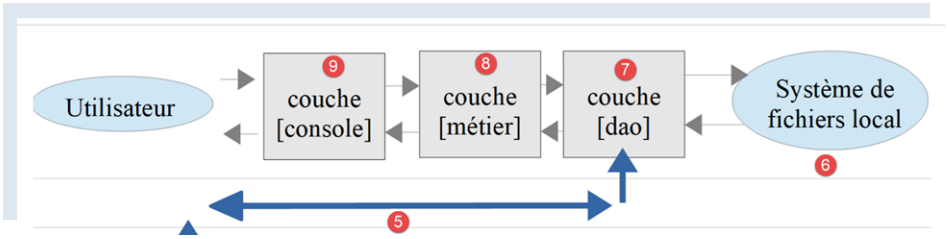
- lignes 9-24 : on crée un environnement d'exécution identique à celui du script serveur **[impots-server]** ;
- ligne 38 : pour construire la couche **[dao]**, on instancie la classe **[ServerDaoWithSession]** ;

Le résultat des tests est le suivant :

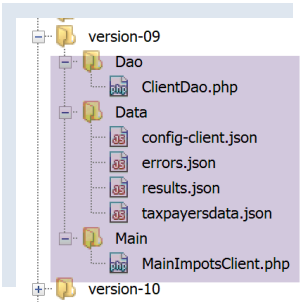


1.19.2 Le client

Nous nous intéressons à la partie cliente de l'application.



Cette architecture sera implémentée par les scripts suivants :



Dans la nouvelle version, seuls changent :

- le fichier de configuration **[config-client.json]** ;
- la couche **[dao]** du client ;

1.19.2.1 La couche [dao]

La couche **[Dao]** évolue de la façon suivante :

```
<?php

namespace Application;

// dépendances
use \Symfony\Component\HttpClient\HttpClient;

class ClientDao implements InterfaceClientDao {
  // utilisation d'un Trait
  use TraitDao;
  // attributs
  private $urlServer;
  private $user;
  private $sessionCookie;

  // constructeur
  public function __construct(string $urlServer, array $user) {
    $this->urlServer = $urlServer;
    $this->user = $user;
  }

  // calcul de l'impôt
  public function calculerImpot(string $marié, int $enfants, int $salaire): array {
    // cookie de session ?
    if (!$this->sessionCookie) {
      // on crée un client HTTP
      $httpClient = HttpClient::create([
          'auth_basic' => [$this->user["login"], $this->user["passwd"]],
          "verify_peer" => false
      ]);
      // on fait la requête au serveur sans cookie de session
      $response = $httpClient->request('GET', $this->urlServer,
        ["query" => [
            "marié" => $marié,
            "enfants" => $enfants,
            "salaire" => $salaire
          ]
      ]);
    } else {
      // on fait la requête au serveur avec le cookie de session
      // on crée un client HTTP
      $httpClient = HttpClient::create([
          "verify_peer" => false
      ]);
      $response = $httpClient->request('GET', $this->urlServer,
        ["query" => [
            "marié" => $marié,
            "enfants" => $enfants,
            "salaire" => $salaire
          ],
          "headers" => ["Cookie" => $this->sessionCookie]
      ]);
    }
    // on récupère la réponse
    $json = $response->getContent(false);
    $array = \json_decode($json, true);
    $réponse = $array["réponse"];
    // logs
    print "$json=json\n";
    // on récupère le statut de la réponse
    $statusCode = $response->getStatusCode();
    // erreur ?
    if ($statusCode !== 200) {
      // on a une erreur - on lance une exception
      $réponse = ["statut HTTP" => $statusCode] + $réponse;
      $message = \json_encode($réponse, JSON_UNESCAPED_UNICODE);
      throw new ExceptionImpots($message);
    }
    if (!$this->sessionCookie) {
      // on récupère le cookie de session
      $headers = $response->getHeaders();
      if (isset($headers["set-cookie"])) {
        // cookie de session ?
```

```
74            foreach ($headers["set-cookie"] as $cookie) {
75              $match = [];
76              $match = preg_match("/^PHPSESSID=(.+?);/", $cookie, $champs);
77              if ($match) {
78                $this->sessionCookie = "PHPSESSID=" . $champs[1];
79              }
80            }
81          }
82        }
83        // on rend la réponse
84        return $réponse;
85      }
86
87    }
```

Commentaires

La modification de la couche **[dao]** consiste à maintenant gérer une session :

- ligne 14 : le cookie de la session ;
- lignes 25-39 : lors de la 1re requête ce cookie n'existe pas : on fait alors la requête auprès du serveur en envoyant les informations d'authentification (ligne 28) ;
- lignes 40-53 : lors des requêtes suivantes, on a normalement le cookie de session. On n'envoie pas alors les informations d'authentification (lignes 42-44) ;
- lignes 69-82 : la réponse du serveur à la 1re requête va comporter un cookie de session. On le récupère. Ce code a déjà été utilisé et expliqué au paragraphe lien ;
- ligne 78 : le cookie de session récupéré est mémorisé dans l'attribut de classe **[$sessionCookie]** ;

Note : on aurait pu garder l'ancienne version de la couche **[dao]** et faire l'authentification à chaque requête car celle-ci a un coût négligeable. Par souci pédagogique, on a voulu rappeler comment un client HTTP pouvait gérer une session.

1.19.2.2 Le fichier de configuration

Le fichier de configuration jSON évolue de la façon suivante :

```
1   {
2       "rootDirectory": "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-09",
3       "taxPayersDataFileName": "Data/taxpayersdata.json",
4       "resultsFileName": "Data/results.json",
5       "errorsFileName": "Data/errors.json",
6       "dependencies": [
7           "/../version-08/Entities/BaseEntity.php",
8           "/../version-08/Entities/TaxPayerData.php",
9           "/../version-08/Entities/ExceptionImpots.php",
10          "/../version-08/Utilities/Utilitaires.php",
11          "/../version-08/Dao/InterfaceClientDao.php",
12          "/../version-08/Dao/TraitDao.php",
13          "/Dao/ClientDao.php",
14          "/../version-08/Métier/InterfaceClientMetier.php",
15          "/../version-08/Métier/ClientMetier.php"
16      ],
17      "absoluteDependencies": [
18          "C:/myprograms/laragon-lite/www/vendor/autoload.php"
19      ],
20      "user": {
21          "login": "admin",
22          "passwd": "admin"
23      },
24      "urlServer": "https://localhost:443/php7/scripts-web/impots/version-09/impots-server.php"
25  }
```

Seule l'URL de la ligne 24 change.

1.19.3 Quelques tests

1.19.3.1 Test 1

Tout d'abord nous exécutons le client dans un environnement sans erreurs. Les résultats sont toujours ceux des versions précédentes. Mais maintenant côté serveur, on a un fichier de logs **[logs.txt]** :

```
1   04/07/19 13:16:08:523 :
2   ---nouvelle requête
```

```
3   04/07/19 13:16:08:529 : Autentification en cours…
4   04/07/19 13:16:08:529 : Authentification réussie [admin, admin]
5   04/07/19 13:16:08:529 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>55555] valides
6   04/07/19 13:16:08:529 : tranches d'impôts prises en base de données
7   04/07/19 13:16:08:534 : {"réponse":{"impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.14}}
8   04/07/19 13:16:08:643 :
9   ---nouvelle requête
10  04/07/19 13:16:08:648 : Authentification prise en session…
11  04/07/19 13:16:08:648 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>50000] valides
12  04/07/19 13:16:08:648 : tranches d'impôts prises en session
13  04/07/19 13:16:08:648 : {"réponse":{"impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.14}}
14  04/07/19 13:16:08:769 :
15  ---nouvelle requête
16  04/07/19 13:16:08:775 : Authentification prise en session…
17  04/07/19 13:16:08:775 : paramètres ['marié'=>oui, 'enfants'=>3, 'salaire'=>50000] valides
18  04/07/19 13:16:08:775 : tranches d'impôts prises en session
19  04/07/19 13:16:08:775 : {"réponse":{"impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14}}
20  04/07/19 13:16:08:888 :
21  ---nouvelle requête
22  …
```

- lignes 3-7 : lors de la 1re requête, il y a authentification et recherche des données en base ;
- lignes 9-14 : lors de la requête suivante, il n'y a plus d'authentification et les données sont prises en session. Cela se répète lors des requêtes suivantes (lignes 15 et au-delà) ;

1.19.3.2 Test 2

Maintenant coupons la base de données MySQL. Côté client, on a le résultat console suivant :

```
1   L'erreur suivante s'est produite : {"statut HTTP":500,"erreur":"SQLSTATE[HY000] [2002] Aucune connexion n'a
    pu être établie car l'ordinateur cible l'a expressément refusée.\r\n"}
2   Terminé
```

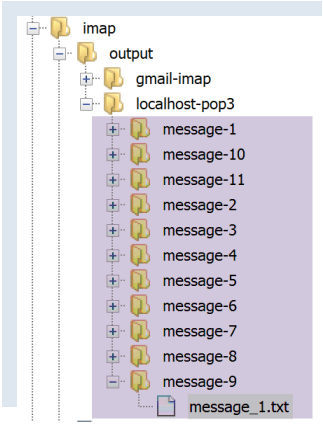
Côté serveur, on a les logs **[logs.txt]** suivant :

```
1   04/07/19 13:19:52:396 :
2   ---nouvelle requête
3   04/07/19 13:19:52:405 : Autentification en cours…
4   04/07/19 13:19:52:405 : Authentification réussie [admin, admin]
5   04/07/19 13:19:52:405 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>55555] valides
6   04/07/19 13:19:52:405 : tranches d'impôts prises en base de données
7   04/07/19 13:19:54:461 : {"réponse":{"erreur":"SQLSTATE[HY000] [2002] Aucune connexion n'a pu être établie
    car l'ordinateur cible l'a expressément refusée.\r\n"}}
8   04/07/19 13:19:55:602 : Message [SQLSTATE[HY000] [2002] Aucune connexion n'a pu être établie car
    l'ordinateur cible l'a expressément refusée.
9   ] envoyé à guest@localhost
10  04/07/19 13:19:55:706 :
11  ---nouvelle requête
12  …
```

Pour avoir le mail reçu par l'administrateur de l'application, on utilise le script **[imap-03.php]** du paragraphe lien avec le fichier de configuration **[config-imap-01.json]** suivant :

```
1.  {
2.    "{localhost:110/pop3}": {
3.      "imap-server": "localhost",
4.      "imap-port": "110",
5.      "user": "guest@localhost",
6.      "password": "guest",
7.      "pop3": "TRUE",
8.      "output-dir": "output/localhost-pop3"
9.    }
10. }
```

On obtient le résultat suivant :

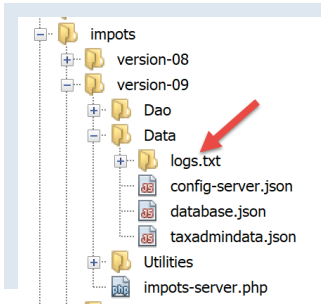


Le fichier **[message_1.txt]** contient le texte suivant :

```
1    return-path: guest@localhost
2    received: from localhost (localhost [127.0.0.1]) by DESKTOP-528I5CU with ESMTP ; Thu, 4 Jul 2019 15:20:22
     +0200
3    message-id: <c82d26df5fb352e10a51577cd1b9ed87@localhost>
4    date: Thu, 04 Jul 2019 13:20:20 +0000
5    subject: plantage du serveur de calcul d'impôts
6    from: guest@localhost
7    to: guest@localhost
8    mime-version: 1.0
9    content-type: text/plain; charset=utf-8
10   content-transfer-encoding: quoted-printable
11
12   SQLSTATE[HY000] [2002] Aucune connexion n'a pu être établie car l'ordinateur cible l'a expressément
     refusée.
```

1.19.3.3 Test 3

Maintenant faison en sorte que le fichier **[logs.txt]** ne puisse être créé. Pour cela, il suffit de créer un dossier **[logs.txt]** :



Ceci fait, exécutons le client.

Côté client, on a les résultats console suivants :

```
1    L'erreur suivante s'est produite : {"statut HTTP":500,"erreur":"Echec lors de la création du fichier de
     logs [Data\/logs.txt]"}
2    Terminé
```

Côté serveur, il n'y a pas de logs mais l'administrateur reçoit le mail suivant :

```
1    return-path: guest@localhost
2    received: from localhost (localhost [127.0.0.1]) by DESKTOP-528I5CU with ESMTP ; Thu, 4 Jul 2019 15:31:49
     +0200
3    message-id: <b2cee274f3437952231d62152ba1cdb3@localhost>
4    date: Thu, 04 Jul 2019 13:31:48 +0000
5    subject: plantage du serveur de calcul d'impôts
6    from: guest@localhost
7    to: guest@localhost
```

```
8   mime-version: 1.0
9   content-type: text/plain; charset=utf-8
10  content-transfer-encoding: quoted-printable
11
12  Echec lors de la création du fichier de logs [Data/logs.txt]
```

1.19.3.4 Test 4

Cette fois-ci, donnons, dans le fichier de configuration du client, des identifiants erronés au client qui se connecte.

Le client affiche les résultats console suivants :

```
1   L'erreur suivante s'est produite : {"statut HTTP":401,"erreur":"Echec de l'authentification [x, x]"}
2   Terminé
```

Côté serveur, les logs suivants apparaissent :

```
1   ---nouvelle requête
2   04/07/19 13:36:05:789 : Autentification en cours…
3   04/07/19 13:36:05:789 : {"réponse":{"erreur":"Echec de l'authentification [x, x]"}}
```

1.19.3.5 Test 5

Remettons le bon utilisateur **[admin, admin]** dans le fichier de configuration du client.

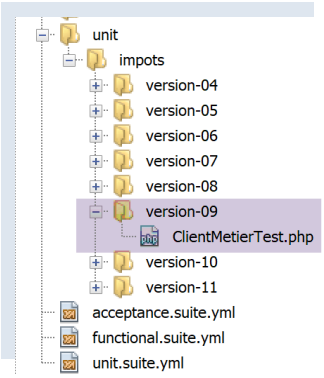
Maintenant demandons l'URL **[http://localhost/php7/scripts-web/impots/version-08/impots-server.php]** du serveur directement dans un navigateur sans passer de paramètres :

Dans le fichier de logs **[logs.txt]** du serveur, on a les lignes suivantes :

```
1   ---nouvelle requête
2   04/07/19 13:37:33:711 : Autentification en cours…
3   04/07/19 13:37:33:711 : Authentification réussie [admin, admin]
4   04/07/19 13:37:33:711 : {"réponse":{"erreurs":["Méthode GET requise avec les seuls paramètres [marié,
    enfants, salaire]","paramètre marié manquant","paramètre enfants manquant","paramètre salaire manquant"]}}
```

1.19.4 Tests [Codeception]

Comme il a été fait pour les versions précédentes, nous allons écrire des tests **[Codeception]** pour la version 09.



1.19.4.1 Test de la couche [métier]

Le test **[ClientMetierTest.php]** est le suivant :

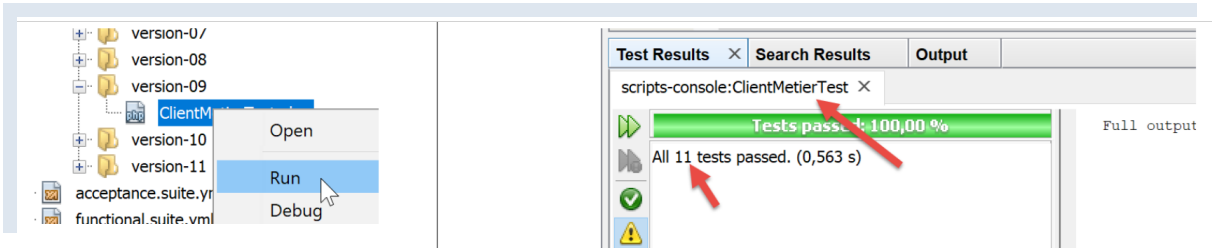
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // définition des constantes
10  define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-09");
```

```
11
12  // chemin du fichier de configuration
13  define("CONFIG_FILENAME", ROOT . "/Data/config-client.json");
14
15  // on récupère la configuration
16  $config = \json_decode(file_get_contents(CONFIG_FILENAME), true);
17
18  // on inclut les dépendances nécessaires au script
19  $rootDirectory = $config["rootDirectory"];
20  foreach ($config["dependencies"] as $dependency) {
21    require "$rootDirectory/$dependency";
22  }
23  // dépendances absolues (bibliothèques tierces)
24  foreach ($config["absoluteDependencies"] as $dependency) {
25    require "$dependency";
26  }
27  //
28  // uses
29  use Codeception\Test\Unit;
30  use const CONFIG_FILENAME;
31  use const ROOT;
32
33  // classe de test
34  class ClientMetierTest extends Unit {
35    …
36  }
```

Commentaires

- par rapport à la classe de test de la version 08, seule change la ligne 10 qui spécifie le dossier racine du client à tester ;

Les résultats du test sont les suivants :



Il est intéressant d'aller voir les logs du serveur **[logs.txt]** :

```
1   04/07/19 13:48:48:525 :
2   ---nouvelle requête
3   04/07/19 13:48:48:536 : Autentification en cours…
4   04/07/19 13:48:48:536 : Authentification réussie [admin, admin]
5   04/07/19 13:48:48:536 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>55555] valides
6   04/07/19 13:48:48:536 : données fiscales prises en base de données
7   04/07/19 13:48:48:548 : {"réponse":{"impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.14}}
8   04/07/19 13:48:48:635 :
9   ---nouvelle requête
10  04/07/19 13:48:48:645 : Autentification en cours…
11  04/07/19 13:48:48:645 : Authentification réussie [admin, admin]
12  04/07/19 13:48:48:645 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>50000] valides
13  04/07/19 13:48:48:645 : données fiscales prises en base de données
14  04/07/19 13:48:48:655 : {"réponse":{"impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.14}}
15  04/07/19 13:48:48:751 :
16  ---nouvelle requête
17  04/07/19 13:48:48:762 : Autentification en cours…
18  04/07/19 13:48:48:762 : Authentification réussie [admin, admin]
19  04/07/19 13:48:48:762 : paramètres ['marié'=>oui, 'enfants'=>3, 'salaire'=>50000] valides
20  04/07/19 13:48:48:762 : données fiscales prises en base de données
21  04/07/19 13:48:48:773 : {"réponse":{"impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14}}
22  04/07/19 13:48:48:865 :
23  ---nouvelle requête
24  …
25  ---nouvelle requête
26  04/07/19 13:48:49:546 : Autentification en cours…
27  04/07/19 13:48:49:546 : Authentification réussie [admin, admin]
28  04/07/19 13:48:49:546 : paramètres ['marié'=>oui, 'enfants'=>3, 'salaire'=>200000] valides
29  04/07/19 13:48:49:546 : données fiscales prises en base de données
```

```
30  04/07/19 13:48:49:551 : {"réponse":{"impôt":42842,"surcôte":17283,"décôte":0,"réduction":0,"taux":0.41}}
```

On constate que les données de l'administration fiscale sont toujours prises dans la base de données et jamais dans la session. Revenons au code du test exécuté :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   …
10
11  // classe de test
12  class ClientMetierTest extends Unit {
13    // couche métier
14    private $métier;
15
16    public function __construct() {
17      parent::__construct();
18      // on récupère la configuration
19      $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
20      // création de la couche [dao]
21      $clientDao = new ClientDao($config["urlServer"], $config["user"]);
22      // création de la couche [métier]
23      $this->métier = new ClientMetier($clientDao);
24    }
25
26    // tests
27    public function test1() {
28      …
29    }
30
31    public function test2() {
32      …
33    }
34
35    public function test3() {
36      …
37    }
38
39    …
40
41  }
```

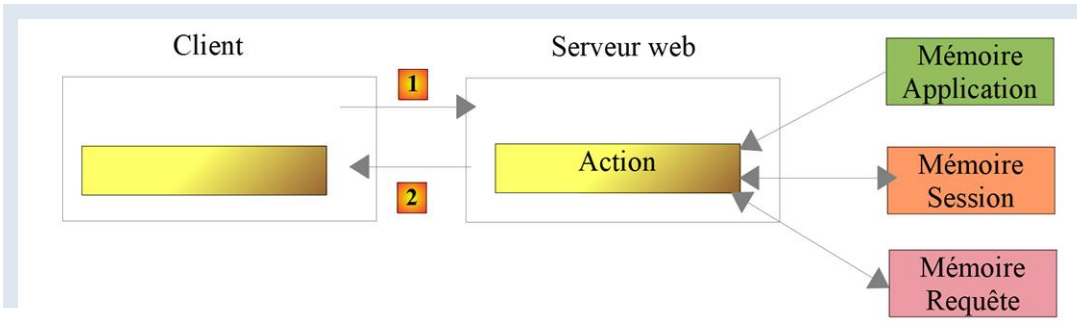
Dans une classe de test **`[Codeception]` le constucteur est exécuté pour chaque test**.

- ligne 21 : un nouveau **`[ClientDao]`** est donc créé pour chaque test avec un cookie de session NULL. Ceci explique que ce client ne profite d'aucune session ;

Cet exemple nous montre que la session n'est pas le bon endroit pour stocker les données de l'administration fiscale. En effet, celles-ci sont communes à tous les utilisateurs de l'application. Or ici, elles sont dupliquées dans chacune des sessions de ceux-ci.

En programmation web, on distingue trois types de visibilité pour les données partagées :

- des données **partagées par tous les utilisateurs** de l'application web. Ce sont en général des données en lecture seule. PHP ne dispose pas nativement de cette mémoire ;
- des données partagées par les requêtes d'**un même client**. Ces données sont mémorisées dans la session. On parle alors de **session client** pour désigner la mémoire du client. Toutes les requêtes d'un client ont accès à cette session. Elles peuvent y stocker et y lire des informations. Dans les scripts précédents, cette session est implémentée par l'objet Symfony **`[HttpFoundation\Session\Session]`** ;
- la **mémoire de requête**, ou contexte de requête. La requête d'un utilisateur peut être traitée par plusieurs actions successives. Le contexte de la requête permet à une action 1 de transmettre de l'information à une action 2. Dans les scripts précédents, la requête est implémentée par l'objet Symfony **`[HttpFoundation\Request]`** et sa mémoire par l'attribut **`[HttpFoundation\Request::attributes]`** ;



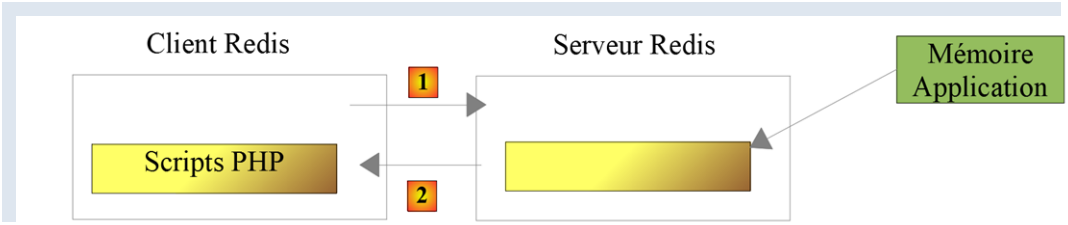
Des bibliothèques tierces existent pour donner à PHP une mémoire d'application. La nouvelle version de l'exercice d'application montre l'usage de l'une d'elles.

1.20 Exercice d'application – version 10

La version précédente a montré que les données fiscales, partagées par tous les utilisateurs de l'application, devraient être stockées dans une mémoire de portée **[Application]**. Nous allons utiliser un serveur Redis **[https://redis.io]** pour implémenter celle-ci.

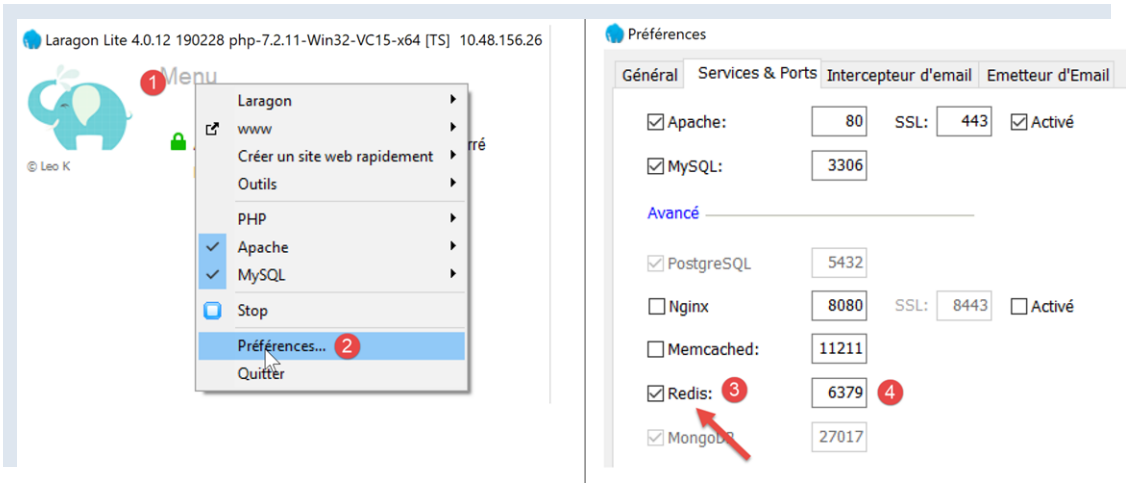
1.20.1 Redis

La mémoire de portée **[Application]** sera implémentée par un serveur Redis. Les scripts PHP ayant besoin de cette mémoire d'application seront des clients de ce serveur :



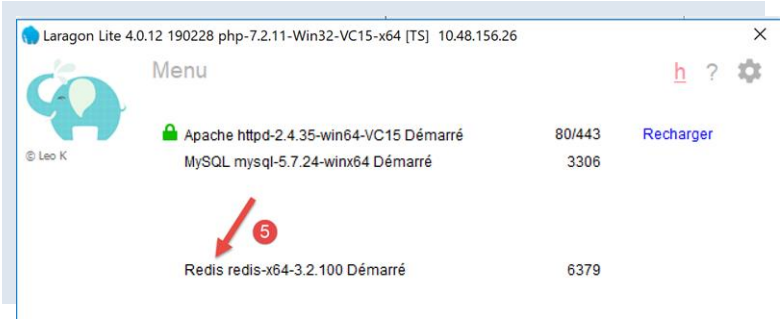
1.20.2 Installation de Redis

Laragon vient avec un serveur Redis non activé par défaut. Il faut donc commencer par l'activer :



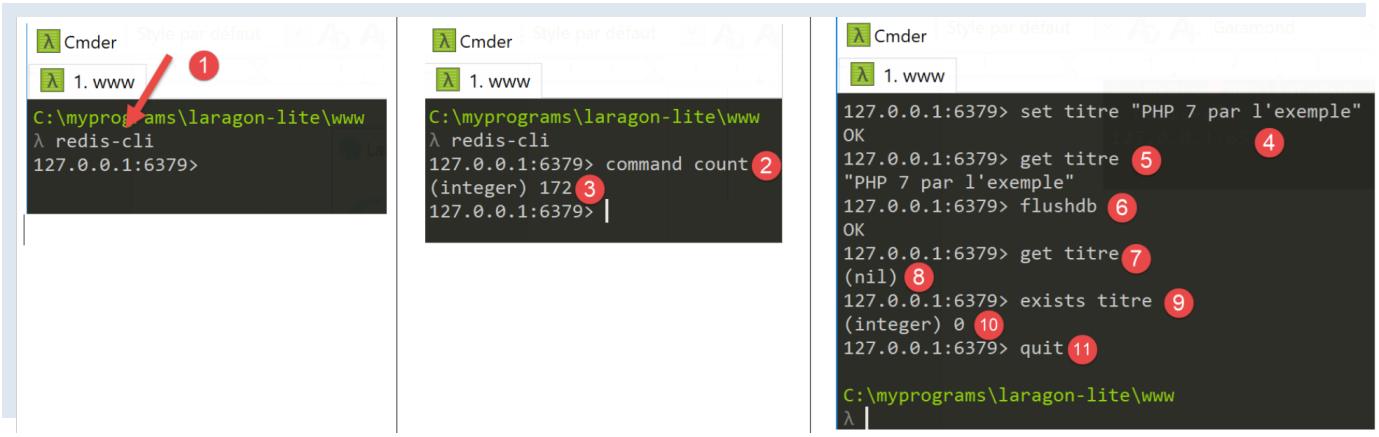
- en **[3]**, activer le serveur **[Redis]** ;
- en **[4]**, laisser le port **[6379]** que les clients Redis utilisent par défaut ;

Les services Laragon sont automatiquement relancés après activation de Redis :



1.20.3 Le client Redis en mode commande

Le serveur Redis peut être interrogé en mode commande. On ouvre un terminal Laragon (cf paragraphe lien) :



- en **[1]**, la commande **[redis-cli]** lance le client en mode commande du serveur Redis ;

En juillet 2019, le client Redis peut utiliser 172 commandes pour dialoguer avec le serveur **[https://redis.io/commands#list]**. L'une d'elles **[command count] [2]**, affiche ce nombre **[3]**.

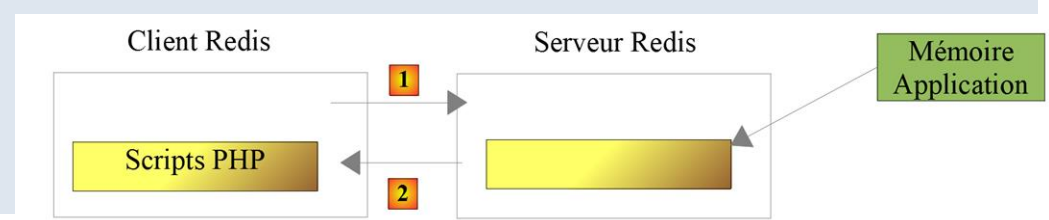
Nous n'allons présenter que celles dont nous allons avoir besoin dans notre application PHP. Nous allons utiliser Redis pour une unique chose : stocker un tableau **['attribut'=>'valeur']** dans la mémoire de Redis. Cela se fait avec la commande Redis **[set attribut valeur] [4]**. La valeur peut ensuite être récupérée avec la commande **[get attribut] [5]**. C'est tout ce dont nous aurons besoin.

Il peut être nécessaire de vider la mémoire de Redis. Cela se fait avec la commande **[flushdb] [6]**. Ensuite si on demande la valeur de l'attribut **[titre] [7]**, on obtient une référence **[nil] [8]** indiquant que l'attribut n'a pas été trouvé. On peut également utiliser la commande **[exists] [9-10]** pour vérifier l'existence d'un attribut.

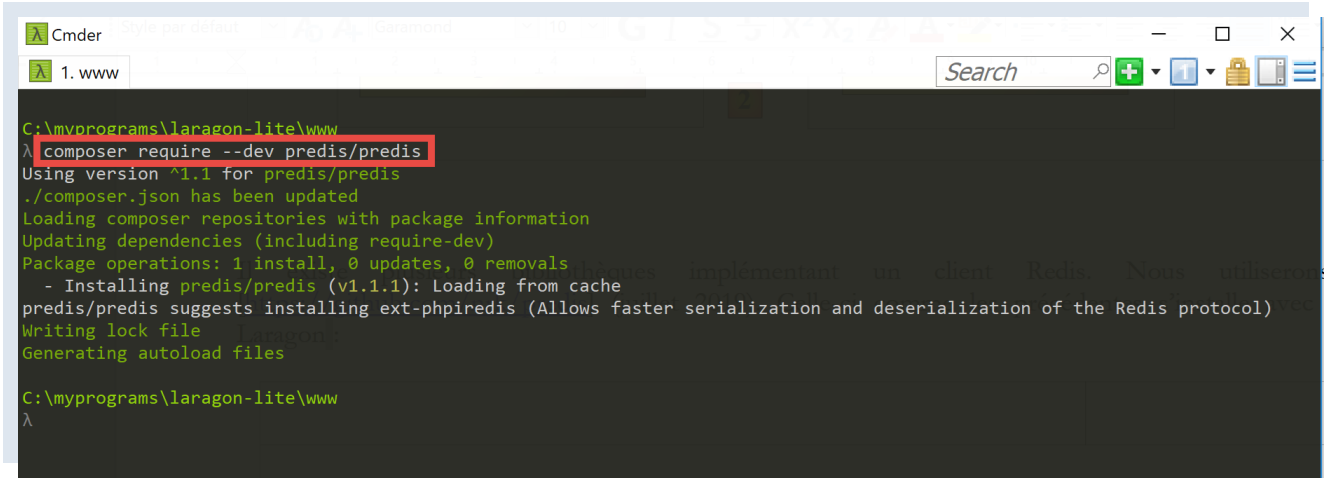
Pour quitter le client Redis, taper la commande **[quit] [11]**.

1.20.4 Installation d'un client Redis pour PHP

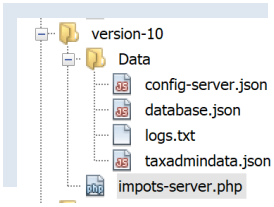
Il nous faut maintenant installer un client Redis pour PHP :



Il existe plusieurs bibliothèques implémentant un client Redis. Nous utiliserons la bibliothèque **[Predis]** **[https://github.com/nrk/predis]** (juillet 2019). Celle-ci comme les précédentes s'installe avec **[composer]** dans un terminal Laragon :



1.20.5 Code du serveur



Le fichier de configuration **[config-server.json]** évolue de la façon suivante :

```
{
    "rootDirectory": "C:/myprograms/Laragon-Lite/www/php7/scripts-web/impots/version-10",
    "databaseFilename": "Data/database.json",
    "relativeDependencies": [
        "/../version-08/Entities/BaseEntity.php",
        "/../version-08/Entities/ExceptionImpots.php",
        "/../version-08/Entities/TaxAdminData.php",
        "/../version-08/Entities/Database.php",
        "/../version-08/Dao/InterfaceServerDao.php",
        "/../version-08/Dao/ServerDao.php",
        "/../version-09/Dao/ServerDaoWithSession.php",
        "/../version-08/Métier/InterfaceServerMetier.php",
        "/../version-08/Métier/ServerMetier.php",
        "/../version-09/Utilities/Logger.php",
        "/../version-09/Utilities/SendAdminMail.php"
    ],
    "absoluteDependencies": [
        "C:/myprograms/Laragon-Lite/www/vendor/autoload.php",
        "C:/myprograms/Laragon-Lite/www/vendor/predis/predis/autoload.php"
    ],
    "users": [
        {
            "login": "admin",
            "passwd": "admin"
        }
    ],
    "adminMail": {
        "smtp-server": "localhost",
        "smtp-port": "25",
        "from": "guest@localhost",
        "to": "guest@localhost",
        "subject": "plantage du serveur de calcul d'impôts",
        "tls": "FALSE",
        "attachments": []
    },
```

```
36      "logsFilename": "Data/logs.txt"
37  }
```

Commentaires

- lignes 5-15 : la version 10 n'amène rien de nouveau en dehors du script **[impots-server.php]**. Elle utilise des éléments des verions 08 et 09 ;
- ligne 19 : une dépendance nécessaire à la bibliothèque **[predis]** que l'on vient d'installer ;

Le code du serveur **[impots-server.php]** évolue de la façon suivante :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // gestion des erreurs par PHP
10  ini_set("display_errors", "0");
11  //
12  // chemin du fichier de configuration
13  define("CONFIG_FILENAME", "Data/config-server.json");
14  // alias de classe
15  use \Application\ServerDaoWithSession as ServerDaoWithRedis;
16
17  // session
18  $session = new Session();
19  $session->start();
20  …
21  …
22  // 1er log
23  $logger->write("\n---nouvelle requête\n");
24
25  // on récupère la requête courante
26  $request = Request::createFromGlobals();
27  // authentification seulement la 1re fois
28  if (!$session->has("user")) {
29    …
30  } else {
31    // log
32    $logger->write("Authentification prise en session…\n");
33  }
34
35  // on a un utilisateur valide - on vérifie les paramètres reçus
36  $erreurs = [];
37  // on doit avoir trois paramètres GET
38  $method = strtolower($request->getMethod());
39  …
40
41  // erreurs ?
42  if ($erreurs) {
43  // on envoie un code d'erreur 400 HTTP_BAD_REQUEST au client
44    sendResponse($response, ["erreurs" => $erreurs], Response::HTTP_BAD_REQUEST, [], $logger);
45    // terminé
46    exit;
47  } else {
48    // logs
49    $logger->write("paramètres ['marié'=>$marié, 'enfants'=>$enfants, 'salaire'=>$salaire] valides\n");
50  }
51
52  // on a tout ce qu'il faut pour travailler
53  // Redis
54  \Predis\Autoloader::register();
55  try {
56    // client [predis]
57    $redis = new \Predis\Client();
58    // on se connecte au serveur pour voir s'il est là
59    $redis->connect();
60  } catch (\Predis\Connection\ConnectionException $ex) {
61    // internal server error
62    doInternalServerError("[redis], " . utf8_encode($ex->getMessage()), $response, $config['adminMail'],
    $logger);
63    // terminé
```

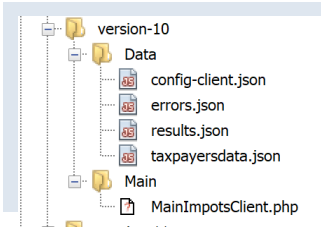
```
64    exit;
65  }
66
67  // création de la couche [dao]
68  if (!$redis->get("taxAdminData")) {
69    // les données fiscales sont prises dans la base de données
70    $logger->write("données fiscales prises en base de données\n");
71    try {
72      // construction de la couche [dao]
73      $dao = new ServerDaoWithRedis($config["databaseFilename"], NULL);
74      // on met les données fiscales dans la mémoire de portée [application]
75      // la méthode [TaxAdminData]->__toString va être appelée implicitement
76      $redis->set("taxAdminData", $dao->getTaxAdminData());
77    } catch (\RuntimeException $ex) {
78      // on note l'erreur
79      doInternalServerError("[dao], " . utf8_encode($ex->getMessage()), $response, $config['adminMail'],
    $logger, $redis);
80      // terminé
81      exit;
82    }
83  } else {
84    // les données fiscales sont prises dans la mémoire de portée [application]
85    $arrayOfAttributes = \json_decode($redis->get("taxAdminData"), true);
86    $taxAdminData = (new TaxAdminData())->setFromArrayOfAttributes($arrayOfAttributes);
87    // isntanciation de la couche [dao]
88    $dao = new ServerDaoWithRedis(NULL, $taxAdminData);
89    // logs
90    $logger->write("données fiscales prises dans redis\n");
91  }
92  // création de la couche [métier]
93  $métier = new ServerMetier($dao);
94  // calcul de l'impôt
95  $result = $métier->calculerImpot($marié, (int) $enfants, (int) $salaire);
96  // on rend la réponse
97  sendResponse($response, $result, Response::HTTP_OK, [], $logger, $redis);
98  // fin
99  exit;
100
101 function doInternalServerError(string $message, Response $response, array $infos,
102   Logger $logger = NULL, \Predis\Client $predisClient = NULL) {
103   // $message : le message d'erreur
104   // $response : réponse HTTP
105   // $infos : tableau d'informations pour l'envoi du mail
106   // $result : tableau des résultats
107   // $logger : le logueur de l'application
108   // $predisClient : un client [predis]
109   //
110   // on envoie un mail à l'administrateur
111   // SendAdminMail intercepte toutes les exception et les logue lui-même
112   $infos['message'] = $message;
113   $sendAdminMail = new SendAdminMail($infos, $logger);
114   $sendAdminMail->send();
115   // on envoie un code d'erreur 500 au client
116   sendResponse($response, ["erreur" => $message], Response::HTTP_INTERNAL_SERVER_ERROR, [], $logger,
    $predisClient);
117 }
118
119 // fonction d'envoi de la réponse HTTP au client
120 function sendResponse(Response $response, array $result, int $statusCode,
121   array $headers, Logger $logger = NULL, \Predis\Client $predisClient = NULL) {
122   // $response : réponse HTTP
123   // $result : tableau des résultats
124   // $statusCode : statut HTTP de la réponse
125   // $headers : entêtes HTTP à mettre dans la réponse
126   // $logger : le logueur de l'application
127   // $predisClient : un client [predis]
128   //
129   // statut HTTTP
130   $response->setStatusCode($statusCode);
131   // body
132   $body = \json_encode(["réponse" => $result], JSON_UNESCAPED_UNICODE);
133   $response->setContent($body);
134   // headers
135   $response->headers->add($headers);
136   // envoi
137   $response->send();
138   // log
```

```
139   if ($logger != NULL) {
140     $logger->write("$body\n");
141     $logger->close();
142   }
143   // fermeture de la connexion [redis]
144   if ($predisClient != NULL) {
145     $predisClient->disconnect();
146   }
147 }
```

Commentaires

- ligne 15 : on donne l'alias **[ServerDaoWithRedis]** à la classe **[\Application\ServerDaoWithSession]** pour refléter le changement d'implémentation du script serveur ;
- lignes 18-19 : la session est conservée. Nous avons ici deux informations à mémoriser :
  - le fait que l'utilisateur se soit authentifié correctement. Cette information est de portée **[session]** : elle est liée à un utilisateur précis et n'est pas valable pour les autres utilisateurs ;
  - les données de l'administration fiscale. Cette information est de portée **[application]** : elle n'est pas liée à un utilisateur précis mais est valable pour tous les utilisateurs ;
- lignes 54-64 : création du client **[redis]** qui va communiquer avec le serveur **[redis]**. Ce client va communiquer avec le port par défaut du serveur. Si celui-ci ne communiquait pas sur son port par défaut ou s'il n'était pas sur la machine **[localhost]**, il faudrait passer ces informations au constructeur de la classe **[\Predis\Client]** ;
- ligne 59 : on connecte tout de suite le client au serveur pour savoir si celui-ci répond ;
- lignes 60-65 : si la connexion au serveur Redis échoue, on envoie une réponse d'erreur au client et un mail sera envoyé à l'administrateur de l'application ;
- ligne 67 : on demande au serveur **[redis]**, la clé **[taxAdminData]**. Si elle n'est pas trouvée, alors les données fiscales sont prises en base de données (ligne 72) ;
- ligne 75 : la clé **[taxAdminData]** est placée dans la mémoire **[redis]** associée à la chaîne jSON de la variable **[$taxAdminData]** qui est un objet de type **[TaxAdminData]**. La méthode **[$redis→set]** s'attend à une chaîne de caractères pour la valeur de la clé. Elle va donc chercher à transformer l'objet de type **[TaxAdminData]** en type **[string]**. C'est alors implicitement, la méthode **[TaxAdminData->__toString]** qui va être appelée. Celle-ci produit la chaîne jSON de l'objet **[TaxAdminData]** ;
- ligne 84 : la clé **[taxAdminData]** est dans la mémoire**[redis]**, alors on récupère sa valeur. On sait que c'est la chaîne jSON d'un objet **[TaxAdminData]**. On décode alors celle-ci pour obtenir un tableau d'attributs ;
- ligne 85 : à partir de ce tableau, un nouvel objet **[TaxAdminData]** est instancié ;
- ligne 87 : la couche **[dao]** est instanciée ;

1.20.6 Code du client



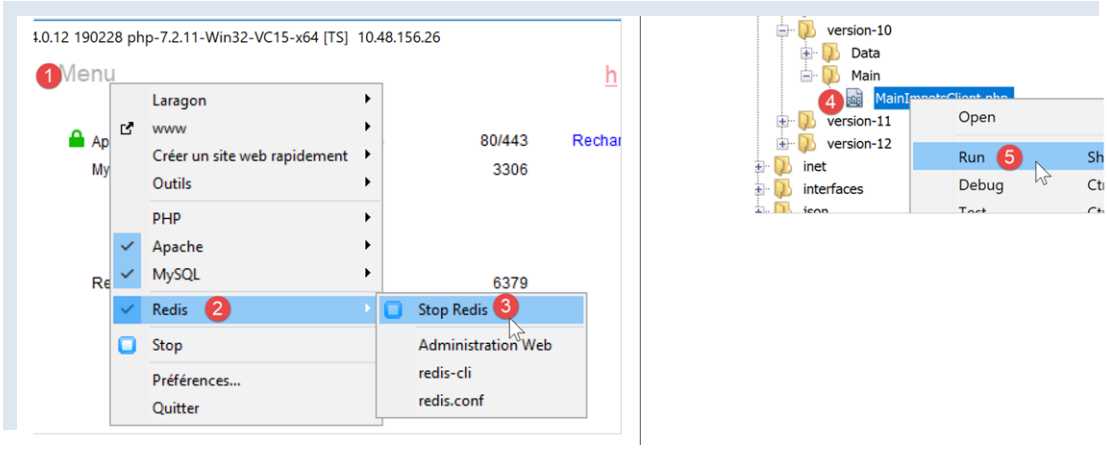
La version 10 du client est identique à la version 9. Seul change le fichier de configuration **[config-client.json]** :

```
1   {
2       "rootDirectory": "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-10",
3       "taxPayersDataFileName": "Data/taxpayersdata.json",
4       "resultsFileName": "Data/results.json",
5       "errorsFileName": "Data/errors.json",
6       "dependencies": [
7           "/../version-08/Entities/BaseEntity.php",
8           "/../version-08/Entities/TaxPayerData.php",
9           "/../version-08/Entities/ExceptionImpots.php",
10          "/../version-08/Utilities/Utilitaires.php",
11          "/../version-08/Dao/InterfaceClientDao.php",
12          "/../version-08/Dao/TraitDao.php",
13          "/../version-09/Dao/ClientDao.php",
14          "/../version-08/Métier/InterfaceClientMetier.php",
15          "/../version-08/Métier/ClientMetier.php"
16      ],
17      "absoluteDependencies": [
18          "C:/myprograms/laragon-lite/www/vendor/autoload.php"
19      ],
```

```
20      "user": {
21          "login": "admin",
22          "passwd": "admin"
23      },
24      "urlServer": "https://localhost:443/php7/scripts-web/impots/version-10/impots-server.php"
25  }
```

Seule change, ligne 24, l'URL du serveur.

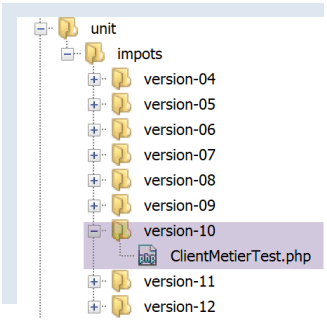
Les résultats sont les mêmes que dans la version 09. Testons simplement un nouveau cas d'erreur :



Le résultat dans la console est le suivant :

```
1   L'erreur suivante s'est produite : {"statut HTTP":500,"erreur":"[redis], Aucune connexion n'a pu être
    établie car l'ordinateur cible l'a expressément refusée. [tcp:\/\/127.0.0.1:6379]"}
2   Terminé
```

1.20.7 Tests [Codeception] du client

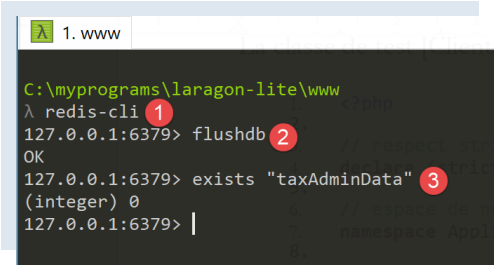


La classe de test **[ClientMetierTest]** de la version 10 est identique à celle de la version 09 à une exception près :

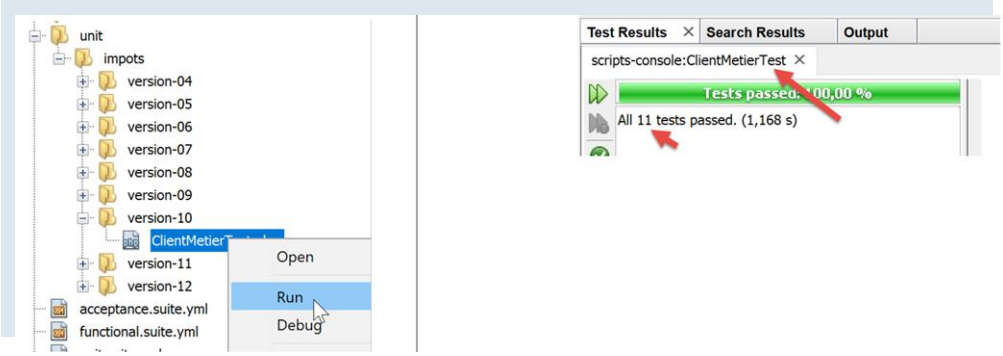
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // définition des constantes
10  define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-10");
11
12  …
13
14  }
```

- ligne 10 : l'environnement du test est celui du client de la version 10 ;

Avant de commencer les tests, supprimons à l'aide du client **[redis-cli]** la clé **[taxAdminData]** de la mémoire du serveur **[redis]** :



Maintenant, exécutons le test :



Maintenant examinons les logs **[logs.txt]** du serveur :

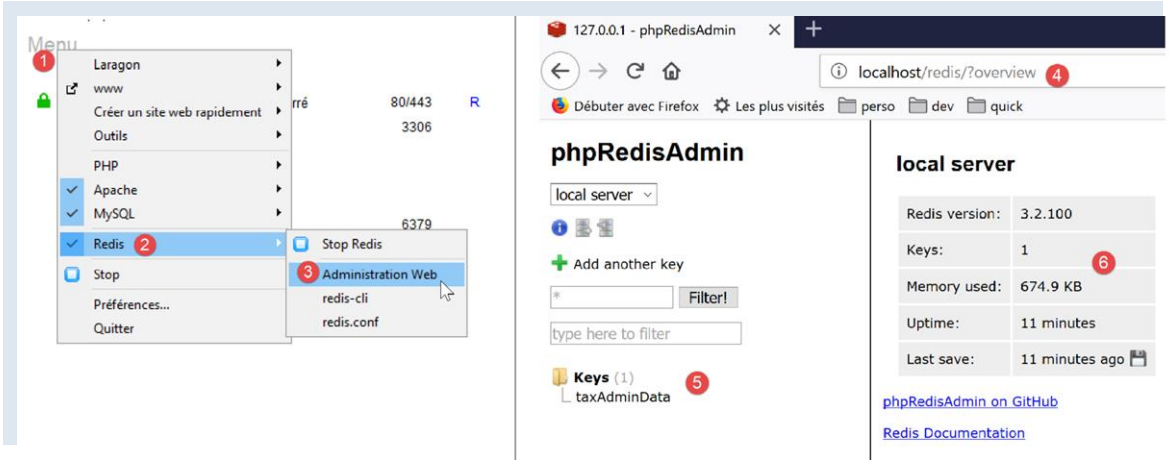
```
1   05/07/19 08:52:16:396 :
2   ---nouvelle requête
3   05/07/19 08:52:16:403 : Autentification en cours…
4   05/07/19 08:52:16:403 : Authentification réussie [admin, admin]
5   05/07/19 08:52:16:403 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>55555] valides
6   05/07/19 08:52:16:407 : données fiscales prises en base de données
7   05/07/19 08:52:16:420 : {"réponse":{"impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.14}}
8   05/07/19 08:52:16:546 :
9   ---nouvelle requête
10  05/07/19 08:52:16:555 : Autentification en cours…
11  05/07/19 08:52:16:555 : Authentification réussie [admin, admin]
12  05/07/19 08:52:16:556 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>50000] valides
13  05/07/19 08:52:16:559 : données fiscales prises dans redis
14  05/07/19 08:52:16:559 : {"réponse":{"impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.14}}
15  05/07/19 08:52:16:668 :
16  ---nouvelle requête
17  05/07/19 08:52:16:675 : Autentification en cours…
18  05/07/19 08:52:16:675 : Authentification réussie [admin, admin]
19  05/07/19 08:52:16:675 : paramètres ['marié'=>oui, 'enfants'=>3, 'salaire'=>50000] valides
20  05/07/19 08:52:16:678 : données fiscales prises dans redis
21  05/07/19 08:52:16:678 : {"réponse":{"impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14}}
22  05/07/19 08:52:16:776 :
23  ---nouvelle requête
24  …
```

On a déjà dit qu'à chaque test, le constructeur de la classe de test est réexécuté, ce qui fait que la classe **[ClientDao]** testée est à chaque test instanciée avec un cookie de session inexistant. Tout se passe donc comme si les 11 tests représentaient 11 utilisateurs différents, avec 11 sessions différentes.

- ligne 6 : les données fiscales sont prises en base de données ;
- lignes 13, 20 : les données fiscales sont prises dans la mémoire **[redis]**. On a donc bien là une mémoire de portée **[application]** partagée par tous les utilisateurs de l'application ;

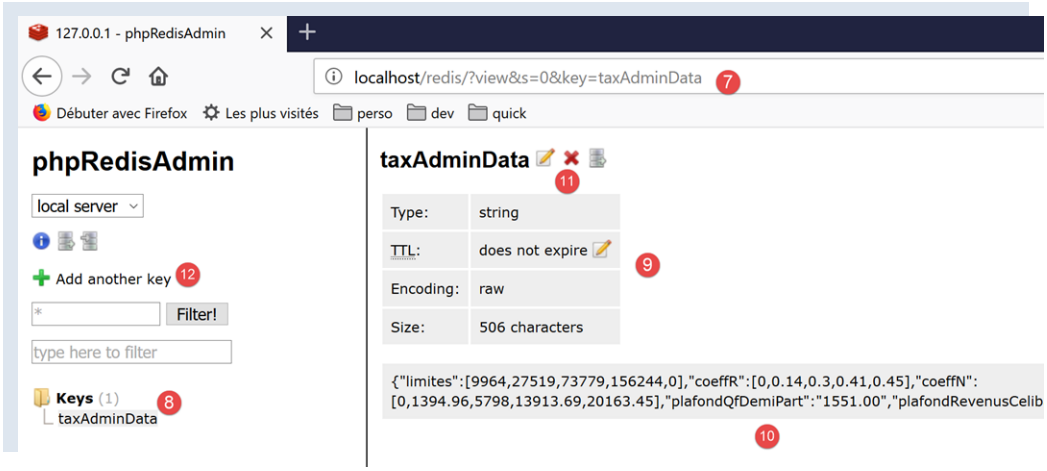
1.20.8 Interface web du serveur [Redis]

Nous avons vu que le serveur **[Redis]** pouvait être géré en mode commande. Il peut également être géré grâce à une interface web :



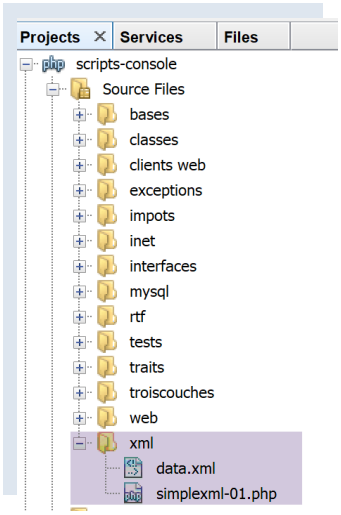
- en **[4]**, l'URL d'administration ;
- en **[5]**, les clés mémorisées par le serveur ;
- en **[6]**, l'état actuel du serveur ;

En cliquant sur **[5]**, on obtient des informations sur la clé **[taxAdminData]** :



- en **[7]**, l'URL qui donne accès aux informations de la clé **[taxAdminData]** **[8]** ;
- en **[9]**, le statut de la clé ;
- en **[10]**, sa valeur : on reconnaît la chaîne jSON d'un objet de type **[TaxAdminData]** ;
- en **[11]**, on peut supprimer la clé ;
- en **[12]**, on peut en ajouter une autre ;

1.21 Traitement de documents XML



Nous considérons le fichier XML **[data.xml]** suivant :

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <tribu>
4       <enseignant>
5           <personne sexe="M">
6               <nom>dupont</nom>
7               <prenom>jean</prenom>
8               <age>28</age>
9               ceci est un commentaire
10          </personne>
11          <section>27</section>
12      </enseignant>
13      <etudiant>
14          <personne sexe="F">
15              <nom>martin</nom>
16              <prenom>charline</prenom>
17              <age>22</age>
18          </personne>
19          <formation>dess IAIE</formation>
20      </etudiant>
21  </tribu>
```

Nous analysons ce document avec le script suivant :

```
1   <?php
2
3   // fichier XML à exploiter
4   $FILE_NAME = "data.xml";
5   // exploitation
6   $xml = simplexml_load_file($FILE_NAME);
7   print_r($xml);
8   print_r($xml->enseignant->personne['sexe']);
9   $nom=$xml->enseignant->personne->nom;
10  print "nom=$nom\n";
11  $sexe=$xml->enseignant->personne['sexe'];
12  print "sexe=$sexe\n";
13  $formation=$xml->etudiant->formation;
14  print "formation=$formation\n";
15  print "isset=".isset($xml->enseignant->personne->nom)."\n";
16  print "isset=".isset($xml->enseignant->personne->xx)."\n";
```

Nous utilisons ici un module PHP appelé **[simpleXML]** qui permet d'exploiter des documents XML.

- ligne 6 : chargement du fichier XML ;
- ligne 7 : affichage du document XML ;
- ligne 8 : affichage de la valeur de l'attribut '*sexe*' d'une personne enseignante : *<enseignant><personne sexe='…'> ;*
- ligne 9 : affichage de la valeur de la 1re balise *<enseignant><personne><nom> ;*

On notera que la balise racine <tribu> n'intervient pas dans le code. Elle pourrait être n'importe quoi ;

Résultats console

```
SimpleXMLElement Object
(
    [enseignant] => SimpleXMLElement Object
        (
            [personne] => SimpleXMLElement Object
                (
                    [@attributes] => Array
                        (
                            [sexe] => M
                        )

                    [nom] => dupont
                    [prenom] => jean
                    [age] => 28
                )

            [section] => 27
        )

    [etudiant] => SimpleXMLElement Object
        (
            [personne] => SimpleXMLElement Object
                (
                    [@attributes] => Array
                        (
                            [sexe] => F
                        )

                    [nom] => martin
                    [prenom] => charline
                    [age] => 22
                )

            [formation] => dess IAIE
        )

)
SimpleXMLElement Object
(
    [0] => M
)
nom=dupont
sexe=M
formation=dess IAIE
isset=1
isset=
```

- lignes 1-37 : le document XML sous la forme d'un objet de type **[simpleXML]** ;

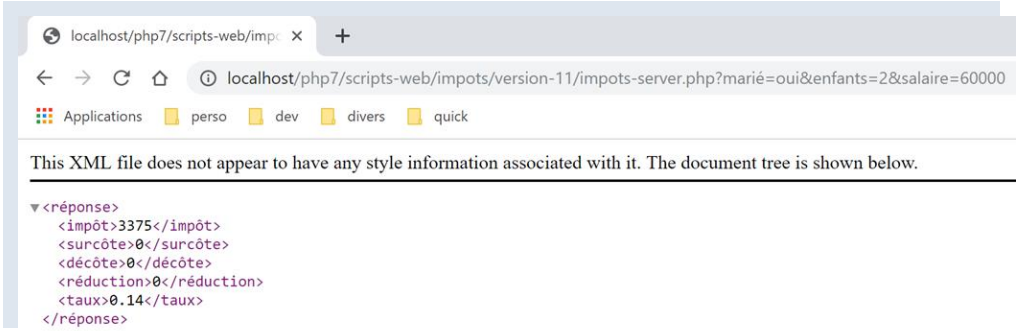
Le script précédent ne nous montre pas toutes les possibilités du module **[simpleXML]** mais il nous suffit pour écrire une nouvelle version de l'exercice d'application.

1.22 Exercice d'application – version 11

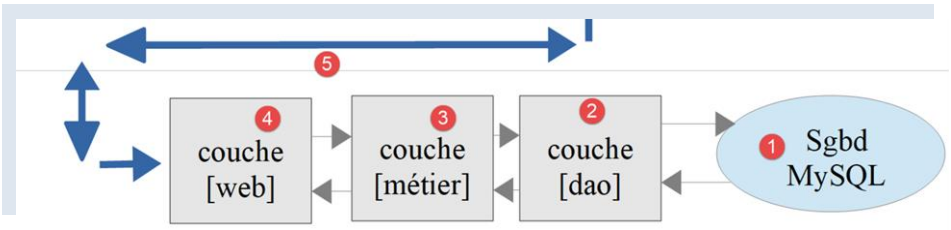
Il est encore fréquent que des services web envoient leur réponse sous la forme d'un flux XML plutôt que d'un flux jSON :

- le flux jSON est plus léger mais il faut un mode d'emploi pour le comprendre ;
- le flux XML est plus verbeux mais il est autodocumenté. Sa compréhension est immédiate ;

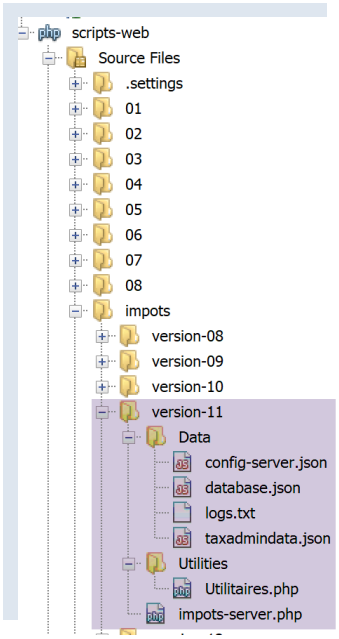
Nous modifions la version 11 client / serveur pour que le serveur envoie désormais un flux XML comme réponse à ses clients :



1.22.1 Le serveur



Cette architecture sera implémentée par les scripts suivants :



1.22.1.1 La classe [Utilitaires]

Nous reprenons la classe **[Utilitaires]** utilisée dès la version 03 (cf paragraphe lien) :

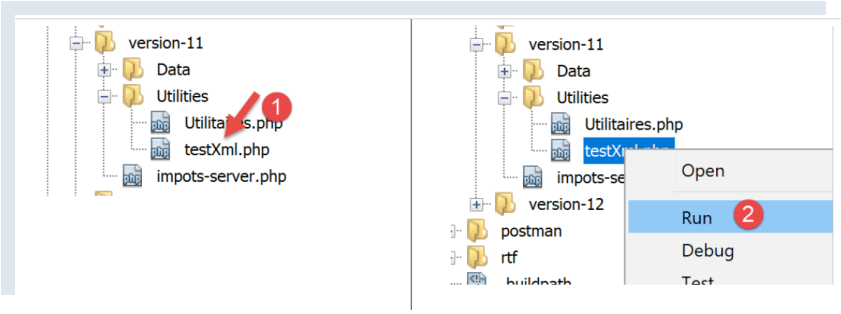
```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   // une classe de fonctions utilitaires
7   abstract class Utilitaires {
8
9     public static function cutNewLinechar(string $ligne): string {
10      …
```

```
11    }
12
13
14    // from https://stackoverflow.com/questions/1397036/how-to-convert-array-to-simplexml
15    public static function getXmlForArrayOfAttributes(array $arrayOfAttributes,
16      \SimpleXmlElement &$node): void {
17      // on scanne les attributs du tableau
18      foreach ($arrayOfAttributes as $attribute => $value) {
19        // l'attribut est-il numérique ?
20        if (is_numeric($attribute)) {
21          // cas de l'index de tableau (mais aussi autres cas)
22          $attribute = 'i' . $attribute;
23        }
24        // $value est-il un tableau ?
25        if (is_array($value)) {
26          // on va explorer le tableau [$value] à son tour
27          // on ajoute un nœud au graphe XML
28          $subnode = $node->addChild($attribute);
29          // appel récursif pour explorer le tableau [$value]
30          Utilitaires::getXmlForArrayOfAttributes($value, $subnode);
31        } else {
32          // on ajoute le nœud au graphe XML
33          $node->addChild("$attribute", htmlspecialchars("$value"));
34        }
35      }
36    }
37
38  }
```

Commentaires

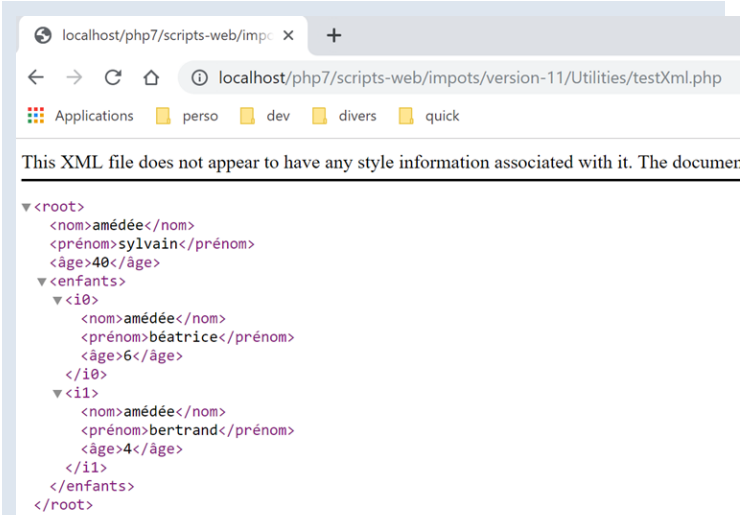
- lignes 14-36 : nous introduisons la méthode statique **[getXmlForArrayOfAttributes]** qui rend la chaîne XML d'un tableau **[arrayOfAttributes]** passé en paramètre. Le second paramètre est la référence d'un nœud d'un graphe XML, de type **[SimpleXmlElement]**. Après exécution, ce nœud contient le graphe XML du tableau **[arrayOfAttributes]** ;

Nous écrivons le test **[testXml.php]** suivant :




```
1   <?php
2
3   // dépendance
4   require __DIR__ . "/Utilitaires.php";
5   // tableau associatif
6   $array = ["nom" => "amédée", "prénom" => "sylvain", "âge" => 40,
7     "enfants" => [["nom" => "amédée", "prénom" => "béatrice", "âge" => 6],
8       ["nom" => "amédée", "prénom" => "bertrand", "âge" => 4]]];
9   // xml
10  header("Content-Type: application/xml");
11  $node = new \SimpleXMLElement("<?xml version='1.0' encoding='UTF-8'?><root></root>");
12  \Application\Utilitaires::getXmlForArrayOfAttributes($array, $node);
13  print $node->asXML();
```

Lorsqu'on exécute ce script **[2]**, nous obtenons la chose suivante dans un navigateur Chrome :



1.22.1.2 Le script serveur

Le script serveur **[impots-server.php]** doit être modifié ainsi que son fichier de configuration **[config-server.json]** :

```
11. {
12.     "rootDirectory": "C:/myprograms/Laragon-Lite/www/php7/scripts-web/impots/version-11",
13.     "databaseFilename": "Data/database.json",
14.     "relativeDependencies": [
15.         "/../version-08/Entities/BaseEntity.php",
16.         "/../version-08/Entities/ExceptionImpots.php",
17.         "/../version-08/Entities/TaxAdminData.php",
18.         "/../version-08/Entities/Database.php",
19.         "/../version-08/Dao/InterfaceServerDao.php",
20.         "/../version-08/Dao/ServerDao.php",
21.         "/../version-09/Dao/ServerDaoWithSession.php",
22.         "/../version-08/Métier/InterfaceServerMetier.php",
23.         "/../version-08/Métier/ServerMetier.php",
24.         "/../version-09/Utilities/Logger.php",
25.         "/../version-09/Utilities/SendAdminMail.php",
26.         "/Utilities/Utilitaires.php"
27.     ],
28.     "absoluteDependencies": [
29.         "C:/myprograms/Laragon-Lite/www/vendor/autoload.php",
30.         "C:/myprograms/Laragon-Lite/www/vendor/predis/predis/autoload.php"
31.     ],
32.     "users": [
33.         {
34.             "login": "admin",
35.             "passwd": "admin"
36.         }
37.     ],
38.     "adminMail": {
39.         "smtp-server": "localhost",
40.         "smtp-port": "25",
41.         "from": "guest@localhost",
42.         "to": "guest@localhost",
43.         "subject": "plantage du serveur de calcul d'impôts",
44.         "tls": "FALSE",
45.         "attachments": []
46.     },
47.     "logsFilename": "Data/logs.txt"
48. }
```

Commentaires

- la racine du projet est désormais le dossier de la version 11 ;
- ligne 16 : on inclut la nouvelle classe **[Utilitaires]** ;

Les modifications du script serveur sont les suivantes :

```php
<?php

// respect strict des types déclarés des paramètres de foctions
declare (strict_types=1);

// espace de noms
namespace Application;

…
// préparation de la réponse JSON du serveur
$response = new Response();
$response->headers->set("content-type", "application/xml");
$response->setCharset("utf-8");
…
// création de la couche [métier]
$métier = new ServerMetier($dao);
// calcul de l'impôt
$result = $métier->calculerImpot($marié, (int) $enfants, (int) $salaire);
// on rend la réponse
sendResponse($response, $result, Response::HTTP_OK, [], $logger, $redis);
// fin
exit;

function doInternalServerError(string $message, Response $response, array $infos,
  …
}

// fonction d'envoi de la réponse HTTP au client
function sendResponse(Response $response, array $result, int $statusCode,
  array $headers, Logger $logger = NULL, \Predis\Client $predisClient = NULL) {
  // $response : réponse HTTP
  // $result : tableau des résultats
  // $statusCode : statut HTTP de la réponse
  // $headers : entêtes HTTP à mettre dans la réponse
  // $logger : le logueur de l'application
  // $predisClient : un client [predis]
  //
  // statut HTTTP
  $response->setStatusCode($statusCode);
  // body XML
  $node = new \SimpleXMLElement("<?xml version='1.0' encoding='UTF-8'?><réponse></réponse>");
  Utilitaires::getXmlForArrayOfAttributes($result, $node);
  $response->setContent($node->asXML());
  // headers
  $response->headers->add($headers);
  // envoi
  $response->send();
  // log
  if ($logger != NULL) {
    // log en jSON
    $log = \json_encode(["réponse" => $result], JSON_UNESCAPED_UNICODE);
    $logger->write("$log\n");
    $logger->close();
  }
  // fermeture de la connexion [redis]
  if ($predisClient != NULL) {
    $predisClient->disconnect();
  }
}
```

Commentaires

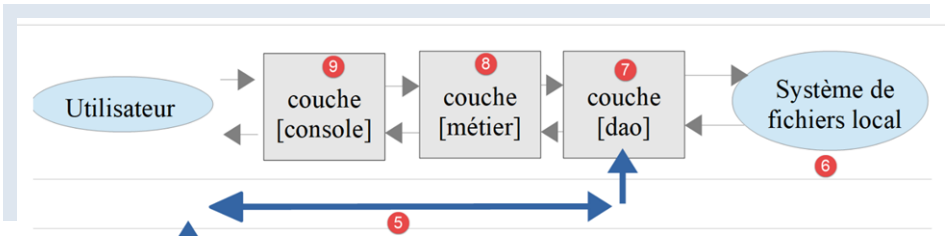
- ligne 12 : on indique que la réponse est de type **[application/xml]** ;
- lignes 29-59 : la réponse du serveur est désormais du XML ;
- ligne 41 : création du nœud racine **[<réponse></réponse>]** du graphe XML ;
- ligne 42 : ce graphe est complété avec le graphe XML du tableau **[$result]** des résultats à envoyer au client ;
- ligne 43 : le graphe XML est converti en chaîne XML pour envoi au client ;

Test

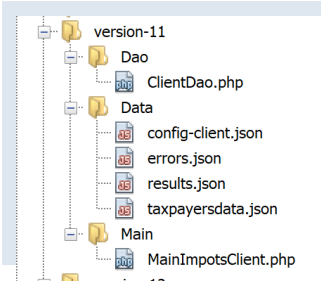
Directement dans un navigateur Chrome, on tape l'URL **[http://localhost/php7/scripts-web/impots/version-11/impots-server.php?mari%C3%A9=oui&enfants=2&salaire=60000]**. On obtient le résultat suivant **[1]** dans un navigateur Chrome :

1.22.2 Le client

Nous nous intéressons maintenant à la partie client de l'application.



Cette architecture sera implémentée par les scripts suivants :



Dans la nouvelle version, seuls changent :

- le fichier de configuration **[config-client.json]** ;
- la couche **[dao]** du client ;

Le fichier de configuration **[config-client.json]** devient le suivant :

```
{
    "rootDirectory": "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-11",
    "taxPayersDataFileName": "Data/taxpayersdata.json",
    "resultsFileName": "Data/results.json",
    "errorsFileName": "Data/errors.json",
    "dependencies": [
        "/../version-08/Entities/BaseEntity.php",
        "/../version-08/Entities/TaxPayerData.php",
        "/../version-08/Entities/ExceptionImpots.php",
        "/../version-08/Utilities/Utilitaires.php",
        "/../version-08/Dao/InterfaceClientDao.php",
        "/../version-08/Dao/TraitDao.php",
        "/Dao/ClientDao.php",
        "/../version-08/Métier/InterfaceClientMetier.php",
        "/../version-08/Métier/ClientMetier.php"
    ],
    "absoluteDependencies": [
        "C:/myprograms/Laragon-Lite/www/vendor/autoload.php"
    ],
```

```
20      "user": {
21          "login": "admin",
22          "passwd": "admin"
23      },
24      "urlServer": "https://localhost:443/php7/scripts-web/impots/version-11/impots-server.php"
25  }
```

1.22.2.1 La couche [dao]

Le client **[ClientDao.php]** (ligne 13 ci-dessus) est modifié pour tenir compte du nouveau format de la réponse. On utilise **[simpleXML]** pour traiter celle-ci :

```
1   <?php
2
3   namespace Application;
4
5   // dépendances
6   use \Symfony\Component\HttpClient\HttpClient;
7
8   class ClientDao implements InterfaceClientDao {
9     // utilisation d'un Trait
10    use TraitDao;
11    // attributs
12    private $urlServer;
13    private $user;
14    private $sessionCookie;
15
16    // constructeur
17    public function __construct(string $urlServer, array $user) {
18      $this->urlServer = $urlServer;
19      $this->user = $user;
20    }
21
22    // calcul de l'impôt
23    public function calculerImpot(string $marié, int $enfants, int $salaire): array {
24      …
25      // on récupère la réponse XML
26      $réponse = $response->getContent(false);
27      $xml = new \SimpleXMLElement($réponse);
28      // logs
29      // print "$réponse\n";
30      // on récupère le statut de la réponse
31      $statusCode = $response->getStatusCode();
32      // erreur ?
33      if ($statusCode !== 200) {
34        // on a une erreur - on lance une exception
35        $message = \json_encode(["statut HTTP" => $statusCode, "réponse" => $xml], JSON_UNESCAPED_UNICODE);
36        throw new ExceptionImpots($message);
37      }
38      if (!$this->sessionCookie) {
39        // on récupère le cookie de session
40        $headers = $response->getHeaders();
41        if (isset($headers["set-cookie"])) {
42          // cookie de session ?
43          foreach ($headers["set-cookie"] as $cookie) {
44            $match = [];
45            $match = preg_match("/^PHPSESSID=(.+?);/", $cookie, $champs);
46            if ($match) {
47              $this->sessionCookie = "PHPSESSID=" . $champs[1];
48            }
49          }
50        }
51      }
52      // on rend la réponse sous forme d'un tableau
53      return \json_decode(\json_encode($xml, JSON_UNESCAPED_UNICODE), true);
54    }
55
56  }
```

Commentaires

- lignes 26-27 : la réponse du serveur est lue. C'est un document XML **[<réponse>…</réponse>]**. Un objet **[SimpleXMLElement]** est construit à partir du document XML reçu ;
- lignes 33-37 : en cas d'erreur, le message de l'exception sera la chaîne jSON de la réponse du serveur plutôt que la chaîne XML recue. En effet, la chaîne jSON est plus concise ;

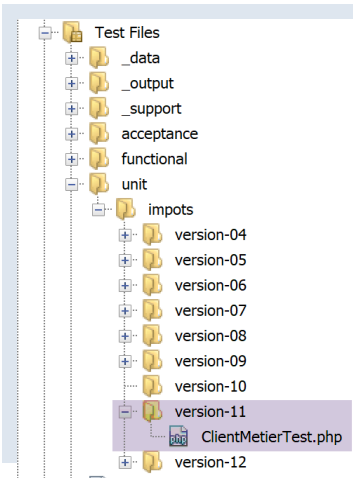
- ligne 53 : on rend le tableau des résultats en deux étapes :
  - l'objet **[$xml]** de type **[\SimpleXMLElement]** est passé en jSON ;
  - on transforme la chaîne jSON obtenue en tableau associatif. C'est le résultat à rendre ;

Test

Si on lance le client avec un environnement correct (base de données, authentification, logs), on obtient les résultats habituels (vérifiez les fichiers **[taxpayersdata.json, results.txt, errors.json]**. Côté serveur, les logs sont eux les suivants :

```
06/07/19 07:41:32:877 :
---nouvelle requête
06/07/19 07:41:32:882 : Autentification en cours…
06/07/19 07:41:32:883 : Authentification réussie [admin, admin]
06/07/19 07:41:32:883 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>55555] valides
06/07/19 07:41:32:908 : données fiscales prises en base de données
06/07/19 07:41:32:959 : {"réponse":{"impôt":2814,"surcôte":0,"décôte":0,"réduction":0,"taux":0.14}}
06/07/19 07:41:33:070 :
---nouvelle requête
06/07/19 07:41:33:077 : Authentification prise en session…
06/07/19 07:41:33:077 : paramètres ['marié'=>oui, 'enfants'=>2, 'salaire'=>50000] valides
06/07/19 07:41:33:099 : données fiscales prises dans redis
06/07/19 07:41:33:100 : {"réponse":{"impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.14}}
06/07/19 07:41:33:189 :
---nouvelle requête
06/07/19 07:41:33:202 : Authentification prise en session…
06/07/19 07:41:33:202 : paramètres ['marié'=>oui, 'enfants'=>3, 'salaire'=>50000] valides
06/07/19 07:41:33:233 : données fiscales prises dans redis
06/07/19 07:41:33:233 : {"réponse":{"impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14}}
06/07/19 07:41:33:318 :
…
```

1.22.2.2 Tests [Codeception]

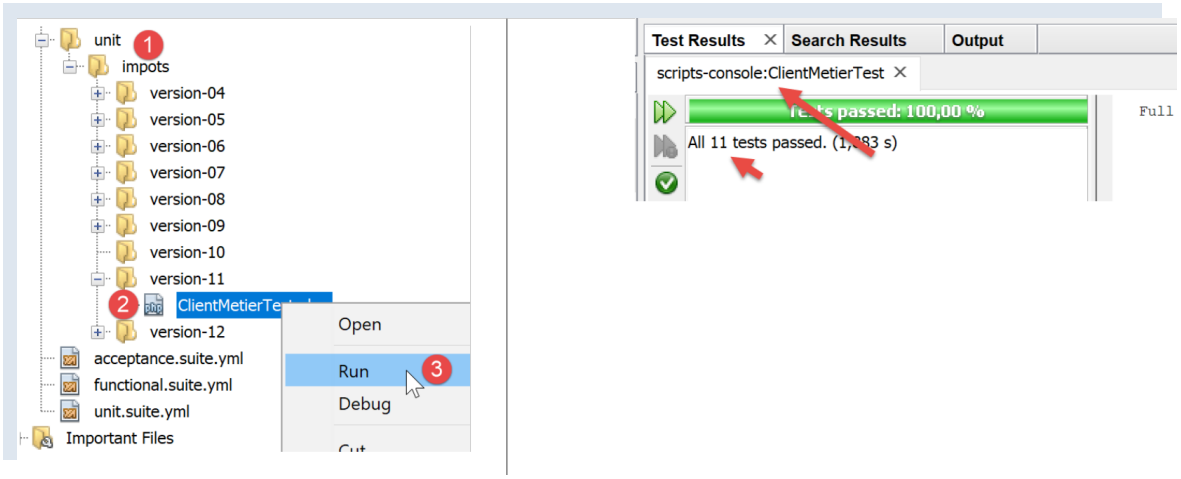


Le test **[ClientMetierTest]** est le suivant :

```
<?php

// respect strict des types déclarés des paramètres de foctions
declare (strict_types=1);

// espace de noms
namespace Application;

// définition des constantes
define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-11");

// chemin du fichier de configuration
define("CONFIG_FILENAME", ROOT . "/Data/config-client.json");

// on récupère la configuration
$config = \json_decode(file_get_contents(CONFIG_FILENAME), true);

```

```php
…
// classe de test
class ClientMetierTest extends Unit {
  // couche métier
  private $métier;

  public function __construct() {
    parent::__construct();
    // on récupère la configuration
    $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
    // création de la couche [dao]
    $clientDao = new ClientDao($config["urlServer"], $config["user"]);
    // création de la couche [métier]
    $this->métier = new ClientMetier($clientDao);
  }

  // tests
  …
}
```

Les résultats du test sont les suivants :

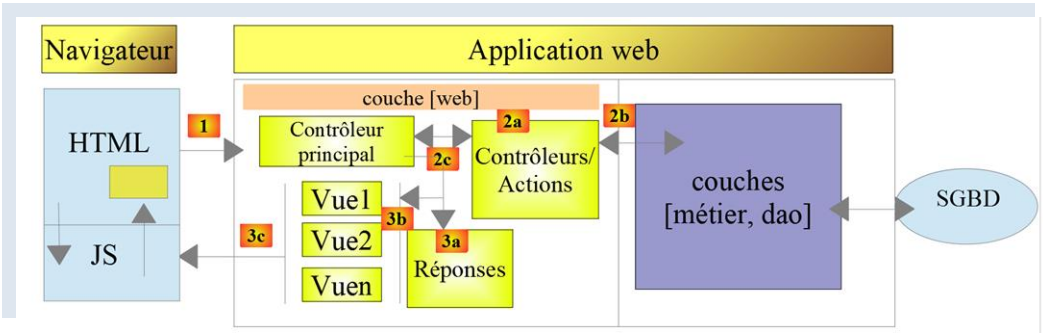


1.23 Exercice d'application – version 12

Nous allons dans ce chapitre écrire une application web respectant l'architecture MVC (Modèle-Vue-Contrôleur). L'application pourra délivrer ses réponses dans trois formats : jSON, XML, HTML. Il y a un saut de complexité entre ce que nous allons faire maintenant et ce qui a été fait précédemment. Nous allons réutiliser la plupart des concepts vus jusqu'à maintenant et nous allons détailler toutes les étapes menant à l'application finale.

1.23.1 Architecture MVC

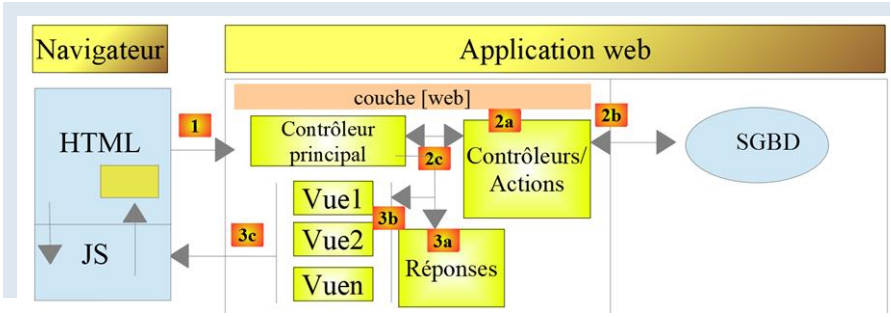
Nous allons implémenter le modèle d'architecture dit MVC (Modèle – Vue – Contrôleur) de la façon suivante :



Le traitement d'une demande d'un client se déroulera de la façon suivante :

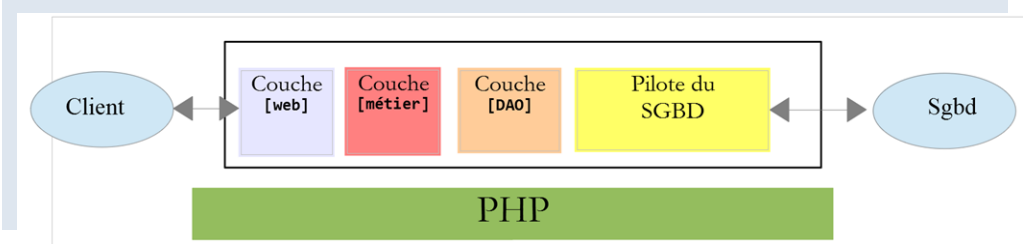
- **1 - demande**
  Les URL demandées seront de la forme *http://machine:port/contexte/….?action=uneAction&param1=v1&param2=v2&…* Le **[Contrôleur principal]** utilisera un fichier de configuration pour " router " la demande vers le bon contrôleur et la bonne action au sein de ce contrôleur. Pour cela, il utilisera le champ **[action]** de l'URL. Le reste de l'URL **[param1=v1&param2=v2&…]** est formé de paramètres facultatifs qui seront transmis à l'action. Le **C** de MVC est ici la chaîne **[Contrôleur principal, Contrôleur / Action]**. Si aucun contrôleur ne peut traiter l'action demandée, le serveur web répondra que l'URL demandée n'a pas été trouvée.
- **2 - traitement**
  - l'action choisie **[2a]** peut exploiter les paramètres *parami* que le **[Contrôleur principal]** lui a transmis. Ceux-ci peuvent provenir de plusieurs sources :
    - du chemin **[/param1/param2/…]** de l'URL,
    - des paramètres **[param1=v1&param2=v2]** de l'URL,
    - de paramètres postés par le navigateur avec sa demande ;
  - dans le traitement de la demande de l'utilisateur, l'action peut avoir besoin de la couche **[métier] [2b]**. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
    - une réponse d'erreur si la demande n'a pu être traitée correctement ;
    - une réponse de confirmation sinon ;
  - le **[Contrôleur / Action]** rendra sa réponse **[2c]** au contrôleur principal ainsi qu'un code d'état. Ces codes d'état représenteront de façon unique l'état dans lequel se trouve l'application. Ce seront soit des codes de réussite, soit des codes d'erreur ;
- **3 - réponse**
  - selon que le client a demandé une réponse jSON, XML ou HTML, le **[Contrôleur principal]** instanciera **[3a]** le type de réponse appropriée et demandera à celle-ci d'envoyer la réponse au client. Le **[Contrôleur principal]** lui transmettra et la réponse et le code d'état fournis par le **[Contrôleur / Action]** qui a été exécuté ;
  - si la réponse souhaitée est de type jSON ou XML, la réponse sélectionnée mettra en forme la réponse du **[Contrôleur / Action]** qu'on lui a donnée et l'enverra **[3c]**. Le client capable d'exploiter cette réponse peut être un script console PHP ou un script Javascript logé dans une page HTML ;
  - si la réponse souhaitée est de type HTML, la réponse sélectionnée sélectionnera **[3b]** une des **vues** HTML **[Vuei]** à l'aide du code d'état qu'on lui a donné. C'est le **V** de MVC. A un code d'état correspond une unique vue. Cette vue **V** va afficher la réponse du **[Contrôleur / Action]** qui a été exécuté. Elle habille avec du HTML, CSS, Javascript les données de cette réponse. On appelle ces données le **modèle de la vue**. C'est le **M** de MVC. Le client est alors le plus souvent un navigateur ;

Maintenant, précisons le lien entre architecture web MVC et architecture en couches. Selon la définition qu'on donne au **modèle**, ces deux concepts sont liés ou non. Prenons une application web MVC à une couche :



Ci-dessus, les **[Contrôleur / Action]** intègrent chacun une partie des couches **[métier]** et **[dao]**. Dans la couche **[web]** on a bien une architecture MVC mais l'ensemble de l'application n'a pas une architecture en couches. Ici il n'y a qu'une couche qui fait tout.

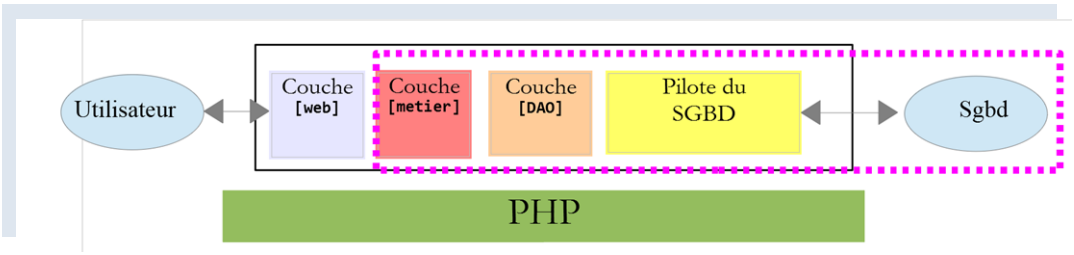
Maintenant, considérons une architecture web multicouche :



La couche **[web]** peut être implémentée sans suivre le modèle MVC. On a bien alors une architecture multicouche mais la couche web n'implémente pas le modèle MVC.

Par exemple, dans le monde .NET la couche **[web]** ci-dessus peut être implémentée avec ASP.NET MVC et on a alors une architecture en couches avec une couche **[web]** de type MVC. Ceci fait, on peut remplacer cette couche ASP.NET MVC par une couche ASP.NET classique (WebForms) tout en gardant le reste (métier, DAO, Pilote) **à l'identique**. On a alors une architecture en couches avec une couche **[web]** qui n'est plus de type MVC.

Dans MVC, nous avons dit que le modèle M était celui de la vue V, c.a.d. l'ensemble des données affichées par la vue V. Une autre définition du modèle M de MVC est donnée :

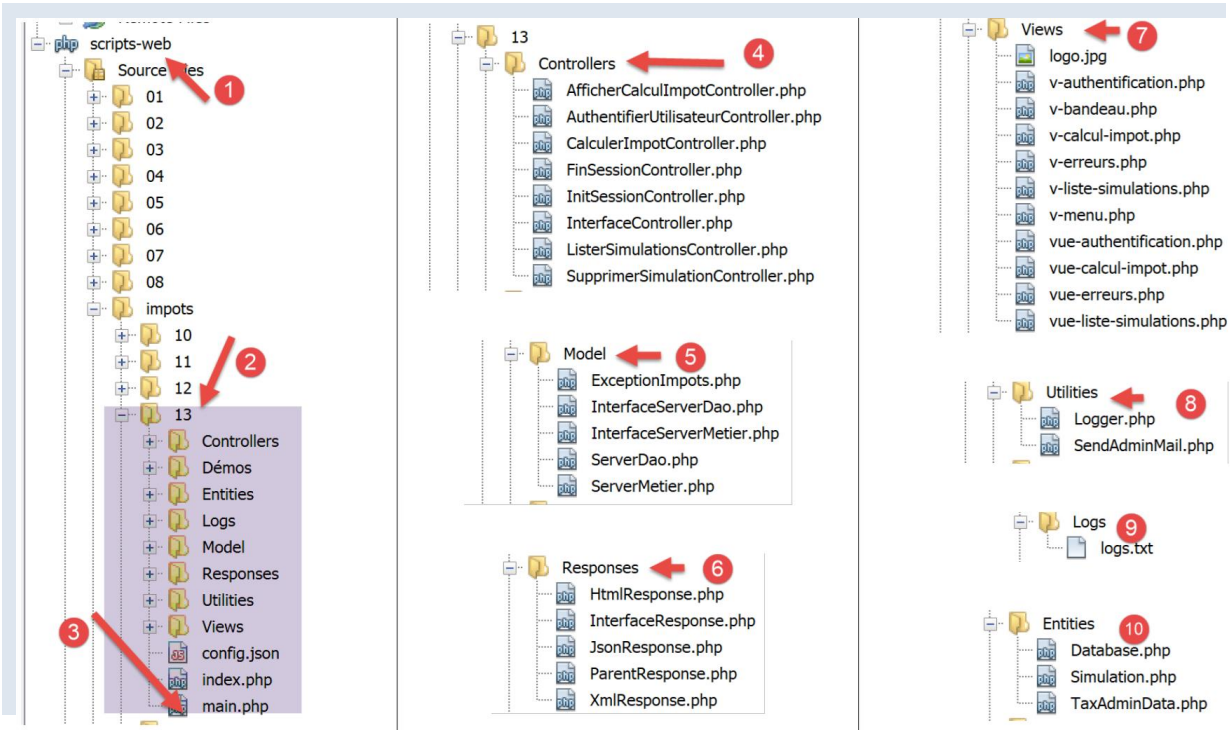


Beaucoup d'auteurs considèrent que ce qui est à droite de la couche **[web]** forme le modèle M du MVC. Pour éviter les ambigüités on peut parler :

- du **modèle du domaine** lorsqu'on désigne tout ce qui est à droite de la couche **[web]** ;
- du **modèle de la vue** lorsqu'on désigne les données affichées par une vue V ;

1.23.2 Arborescence du projet Netbeans

Nous adopterons pour le projet Netbeans une architecture reflétant le modèle MVC :



- **[3]** : **[main.php]** est le contrôleur principal de notre modèle MVC. C'est le C de MVC ;
- **[4]** : le dossier **[Controllers]** contiendra les contrôleurs secondaires. Chacun traite une action particulière. Cette action est indiquée dans l'URL, par exemple **[…/main.php?action=authentifier-utilisateur]**. Avec cette action, le **[Contrôleur principal] [main.php]** va sélectionner un **[Contrôleur secondaire]**, ici **[AuthentifierUtilisateurController]** pour traiter l'action demandée. Ces contrôleurs font aussi partie du C de MVC ;

- **[5]** : le dossier **[Model]** contiendra les couches **[métier]** et **[dao]** de l'application. Selon les termes adoptés précédemment, ces éléments représentent le modèle du domaine et selon la terminologie adoptée pour le M peuvent représenter le M de MVC ;
- **[6]** : le dossier **[Responses]** contient les classes chargées d'envoyer la réponse au client. Il y a une classe par type de réponse souhaitée :
  - **[JsonResponse]** : pour une réponse jSON ;
  - **[XmlResponse]** : pour une réponse XML ;
  - **[HtmlResponse]** : pour une réponse HTML ;
- **[7]** : le dossier **[Views]** contient les vues HTML lorsqu'une réponse HTML est souhaitée. C'est le V de MVC. Elles sont activées par la classe **[HtmlResponse]** qui leur transmet les données à afficher. Ces données sont le modèle de la vue. Selon la terminologie adoptée pour le M, ces données peuvent être le M de MVC ;
- **[8]** : le dossier **[Utilities]** contient des utilitaires :
  - **[Logger]** : la classe qui permet de faire des logs dans un fichier texte ;
  - **[Sendmail]** : la classe qui permet d'envoyer des mails ;
- **[9]** : le dossier **[Logs]** contient le fichier de logs **[logs.txt]** ;
- **[10]** : le dossier **[Entities]** contient des classes utilisées par les différents contrôleurs ;

A l'aide de cette arborescence, on peut décrire le cheminement du traitement d'une action demandée par un client :

- **[main.php] [3]** reçoit la demande ;
- après avoir fait quelques vérifications préliminaires (l'action fait-elle partie des actions acceptées ?), il transmet la demande au contrôleur secondaire **[4]** chargé de traiter cette action ;
- le contrôleur secondaire fait ce qu'il a à faire. Dans son travail, il peut avoir besoin des couches **[métier]** et **[dao] [5]** ainsi que des entités du dossier **[10]**. Il rend sa réponse au contrôleur principal **[main.php]** qui l'a activé ;
- selon le type de réponse **[jSON, XML, HTML]** souhaité par le client, le contrôleur principal **[main.php]** active l'une des réponses du dossier **[Responses] [6]** ;
- les réponses **[JsonResponse, XmlResponse]** envoient respectivement la réponse jSON ou XML au client ;
- la réponse **[HtmlResponse]** utilise l'une des vues du dossier **[Views] [7]** pour envoyer une réponse HTML au client ;
- les différents contrôleurs ont accès à la classe **[Logger]** du dossier **[8]** pour écrire des logs dans le fichier des logs du dossier **[9]**. Sont logués :
  - l'action demandée ;
  - la réponse de son contrôleur. Celle-ci est enregistrée au format jSON quelque soit le type **[jSON, XML, HTML]** demandé ;
- lors d'une erreur fatale (HTTP_INTERNAL_SERVER_ERROR), le contrôleur principal **[main.php]** envoie un mail à l'administrateur à l'aide de la classe **[SendMail]** du dossier **[8]** ;

1.23.3 Les actions de l'application

Le client transmet au serveur web l'action à exécuter sous la forme d'un paramètre **[action]** dans l'URL **[/main.php?action=xxx]**. Les actions autorisées sont listées dans le fichier **[config.json]** qui configure le contrôleur principal **[main.php]** :

```
1   "actions":
2               {
3                   "init-session": "\\InitSessionController",
4                   "authentifier-utilisateur": "\\AuthentifierUtilisateurController",
5                   "calculer-impot": "\\CalculerImpotController",
6                   "lister-simulations": "\\ListerSimulationsController",
7                   "supprimer-simulation": "\\SupprimerSimulationController",
8                   "fin-session": "\\FinSessionController",
9                   "afficher-calcul-impot": "\\AfficherCalculImpotController"
10  },
```

- ligne 1 : la clé **[actions]** du dictionnaire jSON ;
- lignes 3-9 : un dictionnaire **[action:contrôleur]**. A chaque action est associé le contrôleur secondaire chargé de la traiter ;
- ligne 3 : **[init-session]** : démarre une session de simulations de calculs d'impôts. Cette action indique le type de réponses souhaitées **[jSON, XML, HTML]** ;
- ligne 4 : une fois le type de session fixé, le client devra s'authentifier avec l'action **[authentifier-utilisateur]**. Tant qu'il n'est pas identifié, toutes les autres actions sont interdites à l'exception de **[init-session]** ;
- ligne 5 : une fois identifié, le client pourra faire une série de calculs d'impôt avec l'action **[calculer-impot]** ;
- ligne 6 : à tout moment, le client peut demander à voir la liste des simulations qu'il a faites avec l'action **[lister-simulations]** ;
- ligne 7 : il pourra en supprimer certaines avec l'action **[supprimer-simulation]** ;
- ligne 8 : le client termine sa session de simulations avec l'action **[fin-session]**. A partir de ce moment, il devra s'authentifier de nouveau s'il veut utiliser l'application ;
- ligne 9 : dans l'application HTML, l'action **[afficher-calcul-impot]** demande l'affichage du formulaire permettant le calcul de l'impôt ;

1.23.4 Configuration de l'application web

L'application est configurée par le fichier jSON **[config.json]** suivant :

```
{
    "databaseFilename": "database.json",
    "rootDirectory": "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-12",
    "relativeDependencies": [

        "/Entities/BaseEntity.php",
        "/Entities/Simulation.php",
        "/Entities/Database.php",
        "/Entities/TaxAdminData.php",
        "/Entities/ExceptionImpots.php",

        "/Utilities/Logger.php",
        "/Utilities/SendAdminMail.php",

        "/Model/InterfaceServerDao.php",
        "/Model/ServerDao.php",
        "/Model/ServerDaoWithSession.php",
        "/Model/InterfaceServerMetier.php",
        "/Model/ServerMetier.php",

        "/Responses/InterfaceResponse.php",
        "/Responses/ParentResponse.php",
        "/Responses/JsonResponse.php",
        "/Responses/XmlResponse.php",
        "/Responses/HtmlResponse.php",

        "/Controllers/InterfaceController.php",
        "/Controllers/InitSessionController.php",
        "/Controllers/ListerSimulationsController.php",
        "/Controllers/AuthentifierUtilisateurController.php",
        "/Controllers/CalculerImpotController.php",
        "/Controllers/SupprimerSimulationController.php",
        "/Controllers/FinSessionController.php",
        "/Controllers/AfficherCalculImpotController.php"
    ],
    "absoluteDependencies": [
        "C:/myprograms/laragon-lite/www/vendor/autoload.php",
        "C:/myprograms/laragon-lite/www/vendor/predis/predis/autoload.php"
    ],
    "users": [
        {
            "login": "admin",
            "passwd": "admin"
        }
    ],
    "adminMail": {
        "smtp-server": "localhost",
        "smtp-port": "25",
        "from": "guest@localhost",
        "to": "guest@localhost",
        "subject": "plantage du serveur de calcul d'impôts",
        "tls": "FALSE",
        "attachments": []
    },
    "logsFilename": "Logs/logs.txt",
    "actions":
            {
                "init-session": "\\InitSessionController",
                "authentifier-utilisateur": "\\AuthentifierUtilisateurController",
                "calculer-impot": "\\CalculerImpotController",
                "lister-simulations": "\\ListerSimulationsController",
                "supprimer-simulation": "\\SupprimerSimulationController",
                "fin-session": "\\FinSessionController",
                "afficher-calcul-impot": "\\AfficherCalculImpotController"
            },
    "types": {
        "json": "\\JsonResponse",
        "html": "\\HtmlResponse",
        "xml": "\\XmlResponse"
    },
    "vues": {
        "vue-authentification.php": [700, 221, 400],
```

```
73          "vue-calcul-impot.php": [200, 300, 341, 350, 800],
74          "vue-liste-simulations.php": [500, 600]
75      },
76      "vue-erreurs": "vue-erreurs.php"
77  }
```

Commentaires

- ligne 2 : nom du fichier jSON contenant la configuration de l'accès à la base de données ;
- lignes 3-39 : configuration des dépendances du projet. On liste ici la totalité des scripts PHP de l'arborescence du projet ;
- lignes 40-44 : l'utilisateur autorisé à utiliser l'application ;
- lignes 46-54 : les coordonnées mail de l'administrateur de l'application ;
- ligne 55 : le chemin du fichier des logs ;
- lignes 56-65 : associations **[action => contrôleur secondaire chargé de la traiter]** ;
- lignes 66-70 : associations **[type de réponse => classe Response chargée d'envoyer la réponse au client]** ;
- lignes 71-75 : associations **[vue HTML => tableau des codes d'état menant à cette vue]** ;
- ligne 76 : la vue **[vue-erreurs]** est affichée dans une session HTML à chaque fois qu'il se produit une erreur anormale :
  - une application jSON ou XML est habituellement interrogée avec un client programmé. Celui-ci passe au serveur des paramètres qui peuvent absents ou erronés. L'ensemble des contrôleurs traitent ces cas et renvoient au client des codes d'erreur. Tous les cas d'erreur possibles doivent être traités ;
  - avec une application HTML, c'est un peu différent. Utilisée normalement, l'application web n'utilise qu'une partie des cas d'utilisation possibles des clients jSON et XML. Prenons un exemple : l'action **[calculer-impot]** attend trois paramètres postés (envoyés par un POST) : **[marié, enfants, salaire]**.
    - si on a un client jSON permettant de taper des URL à la main, on peut demander l'action **[calculer-impot]** avec un GET plutôt qu'un POST, ou avec un POST sans aucun paramètre posté alors qu'il en faut trois, etc… Le serveur jSON doit traiter tous ces cas ;
    - avec une application web, l'action **[calculer-impot]** sera demandée à partir d'un formulaire web où aucun des deux cas précédents ne sera possible : l'action **[calculer-impot]** sera demandée avec un POST et les trois paramètres **[marié, enfants, salaire]**. Certains de ces paramètres pourront avoir une valeur incorrecte mais ils seront présents. Cependant, l'utilisateur peut reproduire certaines erreurs en tapant lui-même des URL dans le navigateur. Par sécurité, on doit gérer ce cas ;
    - la vue **[vue-erreurs]** sera affichée à chaque fois qu'un contrôleur secondaire rendra un code d'état incompatible avec l'application web, ç-à-d un code d'état non présent aux lignes 72-74 du fichier de configuration. Nous optons pour cette solution dans un souci pédagogique. Une autre option possible serait de ne rien faire et de se contenter de réafficher la vue actuellement affichée dans le navigateur du client pour que l'utilisateur ait l'impression que le serveur ne répond pas à ses URL fabriquées à la main ;

1.23.5 Installation d'outils et de bibliothèques

1.23.5.1 Postman

[Postman] est l'outil qui va nous permettre d'interroger les différentes URL de notre application web. Il nous permet :

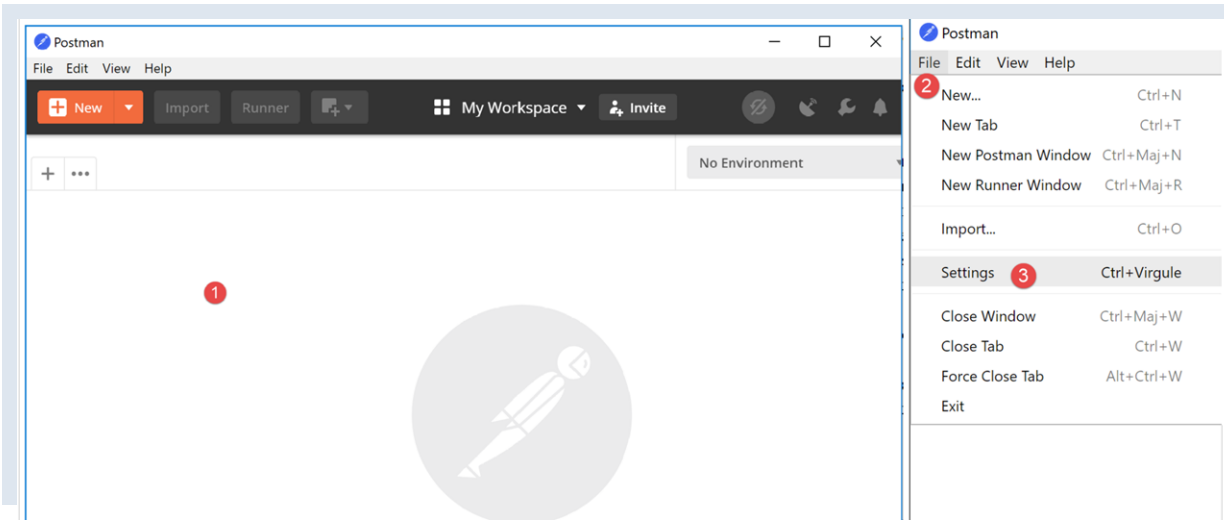
- d'utiliser n'importe quelle URL : celles-ci sont fabriquées à la main ;
- de requêter le serveur web par un GET, POST, PUT, OPTIONS… ;
- de préciser les paramètres du GET ou du POST ;
- de fixer les entêtes HTTP de la requête ;
- de recevoir une réponse au format jSON, XML, HTML,
- d'avoir accès aux entêtes HTTP de la réponse. On a donc ainsi accès à la réponse HTTP complète du serveur ;

Puisque nous fabriquons à la main les URL interrogées, nous allons pouvoir tester tous les cas d'erreur possibles et voir comment le serveur réagit.

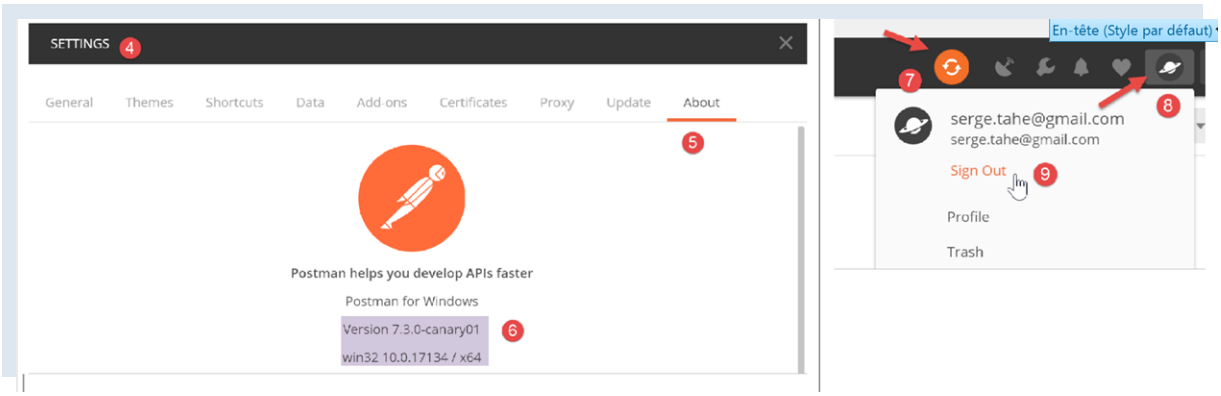
[Postman] est disponible à l'URL **[https://www.getpostman.com/downloads/]**. La version disponible en juin 2019 est la 7.2. Cette version présente une anomalie : lorsqu'on fait des requêtes successives au serveur web interrogé, le client **[Postman 7.2]** ne renvoie pas automatiquement les cookies que le serveur lui envoie, notamment le cookie de session. Pour maintenir la session, il faut alors recopier à la main le cookie de session dans les entêtes HTTP des requêtes successives. Ce n'est pas bien compliqué mais ce n'est pas pratique. C'est un bug qui n'existait pas dans les versions précédentes. Conscient du bug, l'équipe de **[Postman]** l'a corrigé dans une version alpha (peut être instable) appelée **[Postman Canary]** disponible à l'URL **[https://www.getpostman.com/downloads/canary]**. C'est cette version qui est utilisée ici. Nous allons décrire son installation. Si une version stable **[Postman 7.3]** ou ultérieure est disponible, vous pouvez la télécharger : le bug aura probablement été corrigé.

Procédez à l'installation de votre version de **[Postman]**. Au cours de l'installation, on vous demandera de créer un compte : celui-ci sera inutile ici. Le compte **[Postman]** sert à synchroniser différents appareils afin que la configuration de l'un soit répliqué sur un autre. Rien de tout ceci n'est utile ici.

Une fois installé, **[Postman]** présente l'interface suivante :



- en **[2-3]**, on a accès au paramétrage du produit ;



- en **[6]**, la version utilisée dans ce document ;
- si vous avez créé un compte, une synchronisation se fait entre votre poste et un serveur **[Postman]** distant. Cela est symbolisé par la roue **[7]** qui tourne à chaque fois que vous faites des modifications dans le projet **[Postman]**. Pour arrêter cette synchronisation inutile, déconnectez-vous en **[8-9]** ;

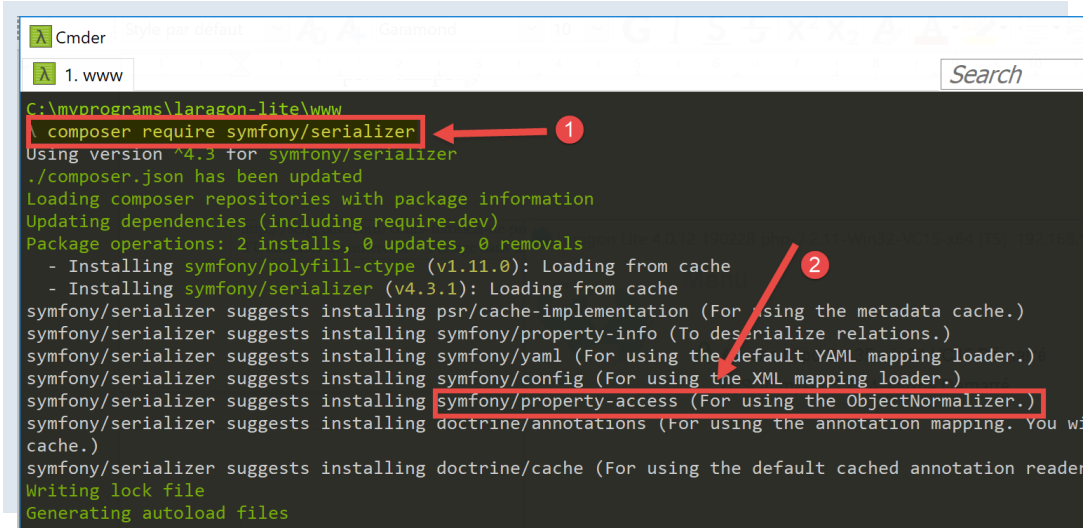
1.23.5.2 La bibliothèque Symfony / Serializer

Pour sérialiser des objets en jSON et XML, nous allons utiliser la bibliothèque **[Symfony / Serializer]**. Elle présente ici deux avantages :

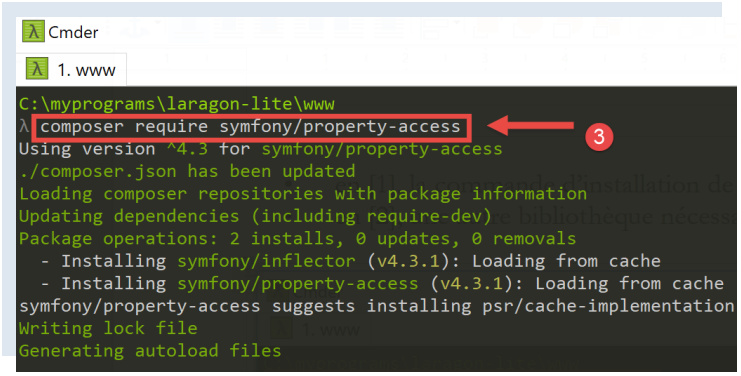
- elle est homogène dans son utilisation pour sérialiser en jSON ou XML : cela évite d'apprendre deux bibliothèques aux API (Application Programming Interface) différentes ;
- nativement, elle sait sérialiser en jSON ou XML des objets, même si les attributs de ceux-ci sont privés. On se rappelle qu'en jSON, pour sérialiser un objet, il fallait que la classe de celui-ci implémente l'interface **[\JsonSerializable]**. Le résultat obtenu alors était la chaîne jSON d'un tableau associatif ayant les attributs de la classe pour clés. Lorsqu'on désérialisait cette chaîne jSON, on retrouvait le tableau associatif primitif, qu'il fallait alors transformer en un objet de la classe qui avait été sérialisée. Avec **[Symfony / Serializer]**, la désérialisation produit tout de suite un objet de la classe sérialisée. C'est plus simple ;

La documentation de la bibliothèque **[Symfony / Serializer]** est disponible à l'URL : **[https://symfony.com/doc/current/components/serializer.html]** (juin 2019).

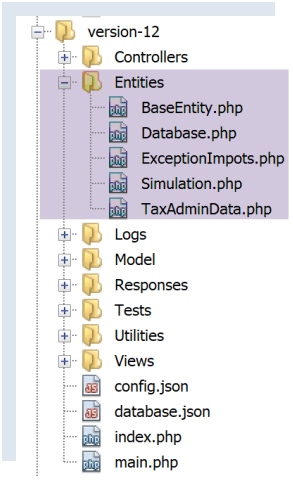
Pour installer cette bibliothèque, ouvrez un terminal Laragon (cf paragraphe lien) et tapez la commande suivante :



- en **[1]**, la commande d'installation de la bibliothèque **[symfony/serializer]** ;
- en **[2]**, une autre bibliothèque nécessaire à notre projet : permet la sérialisation des objets ;



1.23.6 Les entités de l'application



Les entités **[BaseEntity, Database, ExceptionImpots, TaxAdminData]** ont été utilisées dès la version 08 du service web (cf paragraphe lien).

La classe **[Simulation]** va servir à encapsuler les éléments d'une simulation de calcul d'impôt :

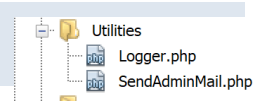
```
1   <?php
2
```

```
3   namespace Application;
4
5   class Simulation extends BaseEntity {
6     // attributs d'une simulation de calcul d'impôt
7     protected $marié;
8     protected $enfants;
9     protected $salaire;
10    protected $impôt;
11    protected $surcôte;
12    protected $décôte;
13    protected $réduction;
14    protected $taux;
15
16    // getters
17    public function getMarié() {
18      return $this->marié;
19    }
20
21    public function getEnfants() {
22      return $this->enfants;
23    }
24
25    public function getSalaire() {
26      return $this->salaire;
27    }
28
29    public function getImpôt() {
30      return $this->impôt;
31    }
32
33    public function getSurcôte() {
34      return $this->surcôte;
35    }
36
37    public function getDécôte() {
38      return $this->décôte;
39    }
40
41    public function getRéduction() {
42      return $this->réduction;
43    }
44
45    public function getTaux() {
46      return $this->taux;
47    }
48
49  }
```

Commentaires

- ligne 5 : la classe **[Simulation]** étend la classe **[BaseEntity]** et hérite donc des méthodes :
  - **[setFromArrayOfAttributes($arrayOfAttributes)]** : qui permet d'initialiser des attributs de la classe ;
  - **[__toString]** : qui rend la chaîne jSON de l'objet ;
- lignes 7-14 : les attributs de la simulation ;
- lignes 16-47 : les getters de la classe ;

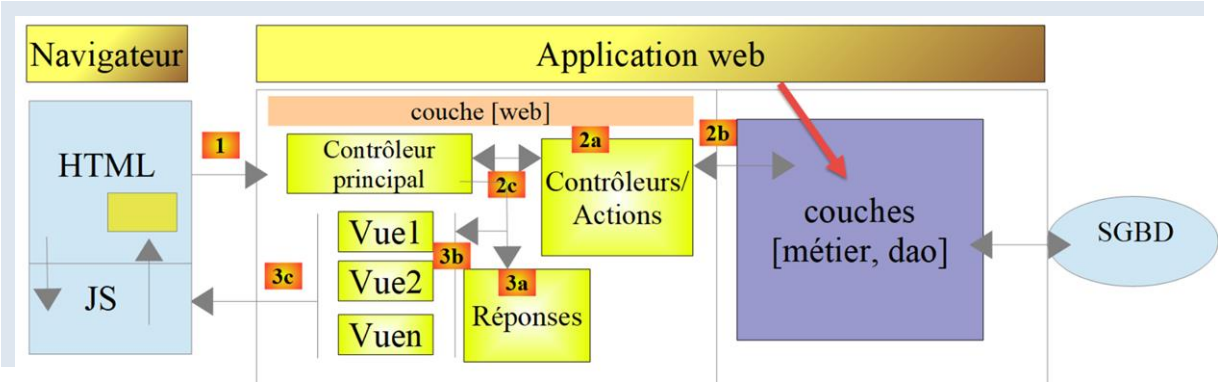
1.23.7 Les Utilitaires de l'application

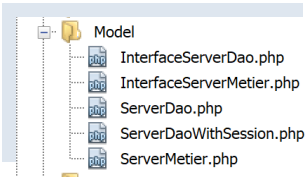


La classe **[Logger]** permet de loguer des événements dans un fichier texte. Cette classe a été décrite au paragraphe lien.

La classe **[SendAdminMail]** permet d'envoyer un mail à l'administrateur de l'application. Cette classe a été décrite au paragraphe lien.

1.23.8 Les couches [métier] et [dao]





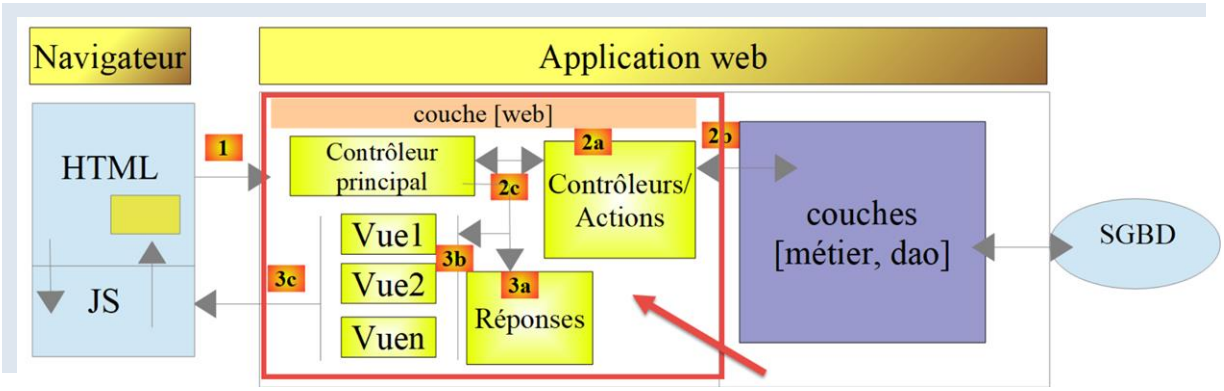
Les classes et interfaces des couches **[métier]** et **[dao]** sont rassemblées dans le dossier **[Model]**. Elles ont toutes été définies et utilisées dans des versions précédentes :

ExceptionImpots	La classe des exceptions lancées par la couche **[dao]**. Définie au paragraphe lien.
InterfaceServerDao	Interface implémentée par la couche **[dao]** du serveur. Définie au paragraphe lien.
ServerDao	Implémentation de l'interface **[InterfaceServerDao]**. Implémente la couche **[dao]** du serveur. Définie au paragraphe lien.
ServerDaoWithSession	Implémentation de l'interface **[InterfaceServerDao]**. Implémente la couche **[dao]** du serveur. Définie au paragraphe lien.
InterfaceServerMetier	Interface implémentée par la couche **[métier]** du serveur. Définie au paragraphe lien.
ServerMetier	Implémentation de l'interface **[InterfaceMetier]**. Implémente la couche **[metier]** du serveur. Définie au paragraphe lien.

L'application en cours d'écriture utilise beaucoup d'éléments déjà présentés et utilisés :

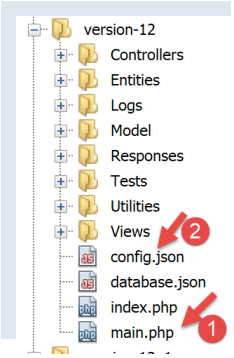
- les couches **[métier]** et **[dao]** ;
- les utilitaires **[Logger]** et **[SendAdminMail]** ;
- les entités **[ExceptionImpots, TaxAdminData, Database]** ;

Nous allons nous concentrer sur la couche **[web]** de l'application :



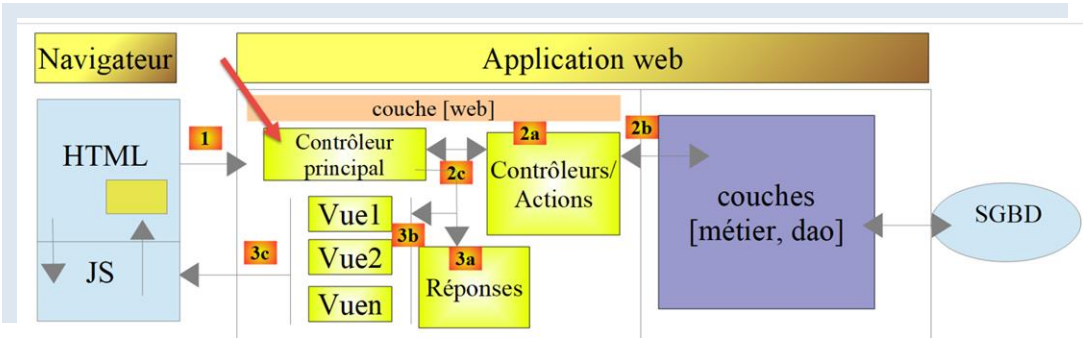
1.23.9 Le contrôleur principal [main.php]

1.23.9.1 Introduction



- **[1-2]** : le contrôleur principal **[main.php] [1]** est configuré par le fichier **[config.json] [2]** ;

Rappelons la position du contrôleur principal dans notre architecture MVC :



En **[1]**, le contrôleur principal **[main.php]** est le 1er élément de l'architecture MVC à traiter la requête du client. Il a plusieurs rôles :

- il fait d'abord les vérifications de base :
  - est-ce que son fichier de configuration existe et est valide ;
  - chargement de toutes les dépendances du projet. Cela revient à charger tous les éléments de l'architecture MVC ;
  - est-ce que l'action demandée a été précisée ? Si oui, est-elle valide ?
  - si l'action demandée est valide, sélectionner **[2a]** le contrôleur secondaire qui va la traiter et lui passer les informations dont il a besoin : la requête HTTP, la session, la configuration de l'application ;
  - récupérer **[2c]** la réponse du contrôleur secondaire. Selon le type (jSON, XML, HTML) d'application demandé par le client, sélectionner **[3a]** la réponse (JsonResponse, XmlResponse, HtmlResponse) chargée d'envoyer la réponse au client et lui passer toutes les informations dont elle a besoin (la requête HTTP, la session, la configuration de l'application, la réponse du contrôleur secondaire) ;
  - une fois cette réponse envoyée **[3c]**, procéder à la libération des ressources qui ont pu être mobilisées pour le traitement de la requête ;

1.23.9.2 [main.php] - 1

Le code du contrôleur principal **[main.php]** est le suivant :

```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // dépendances Symfony
10  use Symfony\Component\HttpFoundation\Request;
11  use Symfony\Component\HttpFoundation\Response;
12  use Symfony\Component\HttpFoundation\Session\Session;
```

```
13
14  // gestion des erreurs par PHP
15  //ini_set("display_errors", "0");
16  error_reporting(E_ALL && !E_WARNING && !E_NOTICE);
17  // on récupère la configuration
18  $configFilename = "config.json";
19  $fileContents = \file_get_contents($configFilename);
20  $erreur = FALSE;
21  // erreur ?
22  if (!$fileContents) {
23    // on note l'erreur
24    $état = 131;
25    $erreur = TRUE;
26    $message = "Le fichier de configuration [$configFilename] n'existe pas";
27  }
28  if (!$erreur) {
29    // on récupère le code JSON du fichier de configuration dans un tableau associatif
30    $config = \json_decode($fileContents, true);
31    // erreur ?
32    if (!$config) {
33      // on note l'erreur
34      $erreur = TRUE;
35      $état = 132;
36      $message = "Le fichier de configuration [$configFilename] n'a pu être exploité correctement";
37    }
38  }
39  // erreur ?
40  if ($erreur) {
41    // préparation de la réponse JSON du serveur
42    // on ne peut pas s'aider du fichier de configuration
43    // dépendances symfony
44    require_once "C:/myprograms/laragon-lite/www/vendor/autoload.php";
45    // préparation réponse
46    $response = new Response();
47    $response->headers->set("content-type", "application/json");
48    $response->setCharset("utf-8");
49    // code de statut
50    $response->setStatusCode(Response::HTTP_INTERNAL_SERVER_ERROR);
51    // contenu
52    $response->setContent(json_encode(["action" => "", "état" => $état, "réponse" => $message],
    JSON_UNESCAPED_UNICODE));
53    // envoi
54    $response->send();
55    // fin
56    exit;
57  }
58  …
```

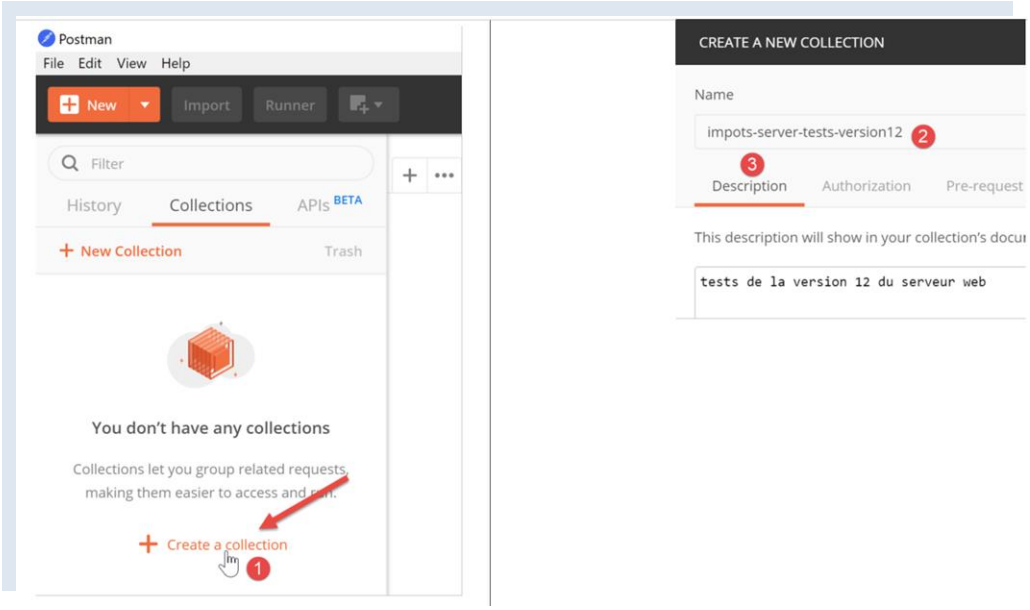
Commentaires

- lignes 10-12 : le contrôleur principal utilise les objets suivants de Symfony :
  - **[Request]** : la requête HTTP en cours de traitement ;
  - **[Session]** : la session de l'application web ;
  - **[Response]** : la réponse HTTP au client ;
- ligne 15 : pendant tout le développement on gardera cette ligne en commentaires : les erreurs PHP sont alors intégrées dans le flux texte envoyé au client. Si ce client est un navigateur, cela permet de voir les erreurs rencontrées par le serveur. C'est une aide au débogage ;
- ligne 16 : toutes les erreurs sont signalées (E_ALL) sauf les avertissements (! E_WARNING) et les informations non fatales (! E_NOTICE). Par exemple, si un fichier ne peut être ouvert, PHP émet une erreur de type **[E_NOTICE]**. Si la ligne 15 permet l'affichage des erreurs, l'erreur d'ouverture du fichier apparaît dans le navigateur client. C'est bien si vous avez oublié de tester le résultat de l'ouverture du fichier, moins bien si vous avez prévu le test : une ligne de **[notice]** vient alors polluer la réponse du serveur au client. En phase de développement, la ligne 16 devrait elle-aussi être commentée : vous ne voulez rater aucune erreur ;
- ligne 19 : le fichier de configuration est lu ;
- lignes 22-27 : si cette lecture s'est mal passée, on note l'erreur (ligne 25), on met l'application dans l'état **[131]** et on prépare un message d'erreur ;
- ligne 30 : on décode la chaîne jSON du fichier de configuration ;
- lignes 32-37 : si ce décodage se passe mal, on note l'erreur (ligne 34), on met l'application dans l'état **[132]** et on prépare un message d'erreur ;
- lignes 40-57 : en cas d'erreur de lecture du fichier de configuration, on ne peut plus avancer. On prépare alors une réponse jSON au client :
- ligne 44 : comme le fichier de configuration n'a pas été lu, il faut importer à la main le fichier **[autoload]** nécessaire à **[Symfony]** ;

- lignes 46-47 : on prépare une réponse jSON ;
- ligne 50 : le code HTTP de la réponse sera 500 INTERNAL_SERVER_ERROR ;
- ligne 52 : on fixe le contenu jSON de la réponse. Toutes les réponses faites par l'application web étudiée auront trois clés :
  - **[action]** : l'action demandée par le client ;
  - **[état]** : l'état de l'application après exécution de cette action ;
  - **[réponse]** : la réponse du serveur web ;
- ligne 54 : la réponse jSON est envoyée au client ;

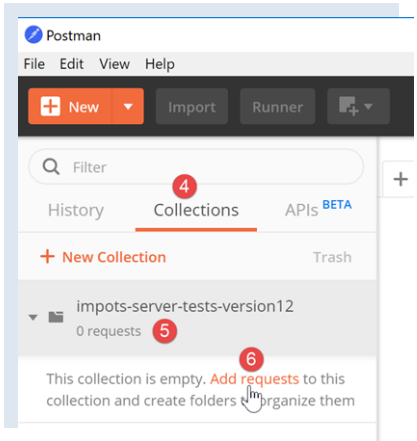
1.23.9.3 Tests [Postman] - 1

Nous allons vérifier le comportement du serveur lorsque fichier de configuration est absent ou incorrect :

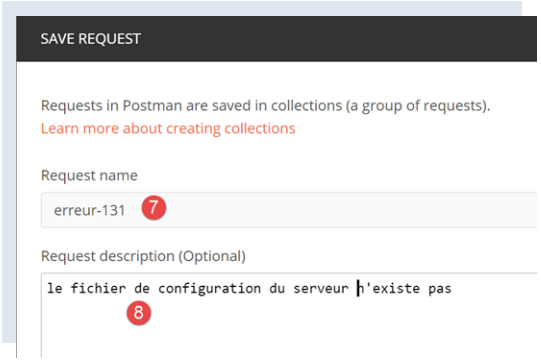


Nous allons rassembler les différentes requêtes que notre client **[Postman]** va émettre vers le serveur d'impôts dans des collections.

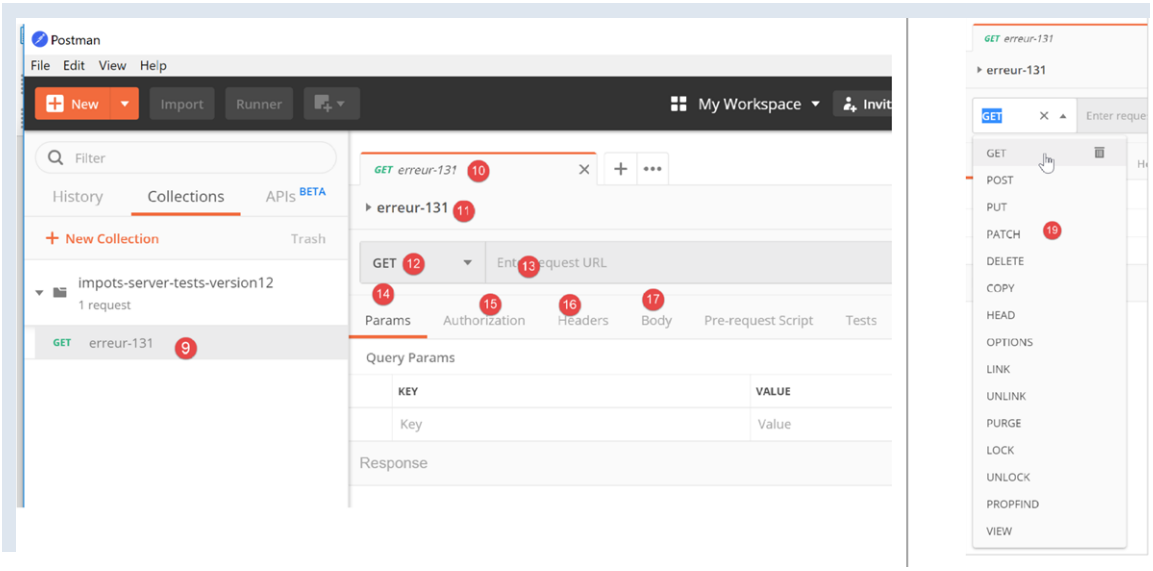
- en **[1]**, créez une nouvelle collection ;
- en **[2]**, donnez-lui un nom ;
- en **[3]**, la description est facultative ;



- dans les collections **[4]**, apparaît maintenant une collection nommée **[impots-server-tests-version12] [5]** ;
- en **[6]**, on peut ajouter une nouvelle requête à la collection ;



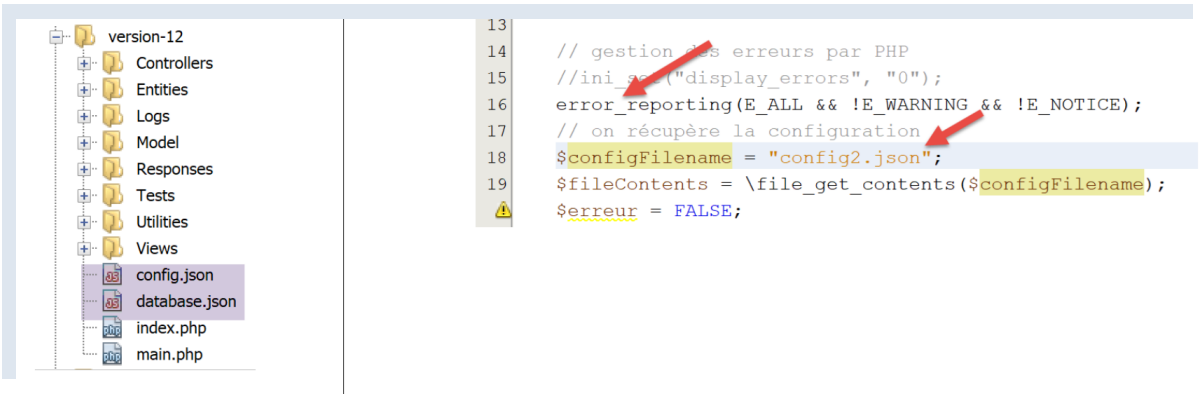
- en **[7]**, on donne un nom à la requête ;
- en **[8]**, la description est facultative ;



- en **[9-11]**, la requête ajoutée à la collection ;
- en **[12]**, choix du type de la requête, ici une requête **[GET]**. En **[19]**, les différents types de requête disponibles ;
- en **[13]**, on tape ici l'URL du serveur ;
- en **[14]**, on met ici les paramètres ajoutés à l'URL et qui seront donc des paramètres du GET. L'intérêt de les mettre ici plutôt que directement dans l'URL est qu'ils seront URL-encodés par **[Postman]**. Si vous les mettez vous-mêmes dans l'URL ce sera à vous de les URL-encoder ;
- en **[15]**, **[Authorization]** sert à définir l'utilisateur qui va se connecter. Nous n'aurons pas à utiliser cette possibilité ;
- en **[16]**, les entêtes HTTP qui accompagneront la requête. Un certain nombre d'entêtes sont automatiquement inclus dans la requête. Vous pouvez ici en ajouter de nouveaux ;
- en **[17]**, **[Body]** désigne les paramètres d'une opération **[POST]**. Nous aurons à utiliser cette option ;

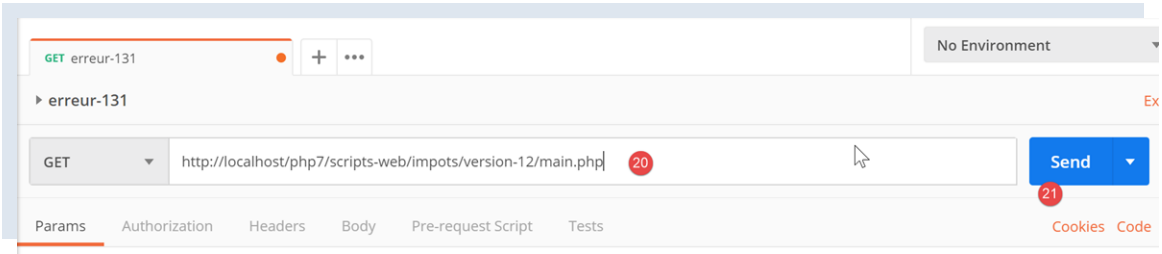
Nous allons faire le test suivant :

- dans **[main.php]**, on indique que le fichier de configuration est **[config2.json]** qui n'existe pas :

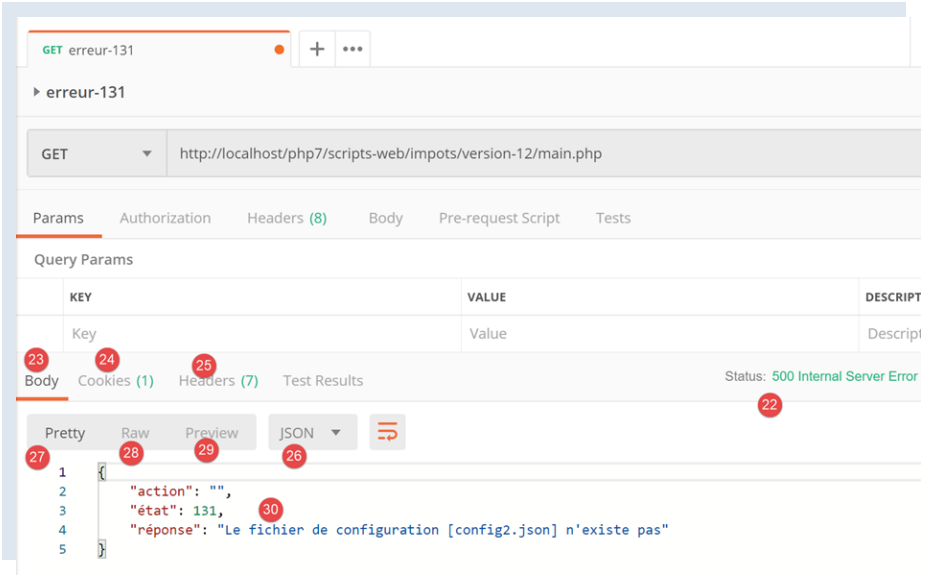


- la ligne 16 du code doit être décommentée ;
- ligne 18 : l'erreur sur le nom du fichier de configuration ;

Entrons dans **[Postman]** **[13, 20]**, l'URL du serveur web de calcul d'impôt et exécutons-la **[21]** :

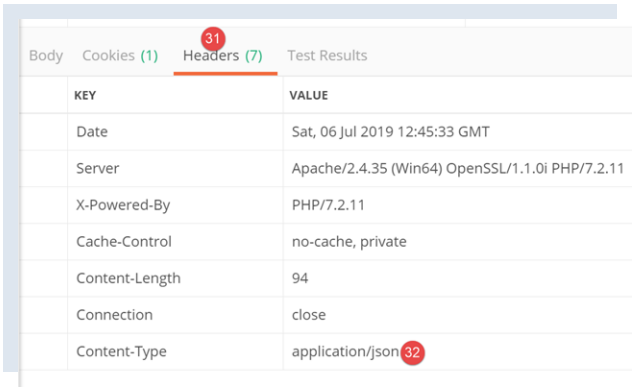


La réponse renvoyée par le serveur (il faut bien sûr que Laragon soit actif) est la suivante :



- en **[22]**, le serveur a renvoyé un code HTTP **[500 Internal Server Error]** ;
- en **[23]**, **[Body]** désigne le corps de la réponse, ç-à-d le document envoyé par le serveur derrière les entêtes HTTP **[28]** ;
- en **[26]**, on voit que **[Postman]** a reçu une réponse jSON ;
- en **[27]**, la réponse jSON mise en forme ;
- en **[28]**, la réponse jSON brute sans mise en forme ;
- en **[29]**, le mode **[Preview]** est utilisé lorsque la réponse est du HTML. Le mode **[Preview]** affiche alors la page reçue ;
- en **[30]**, la réponse jSON du serveur. C'est bien celle que nous attendions ;

En **[25]**, les entêtes HTTP envoyés dans la réponse du serveur sont les suivants :

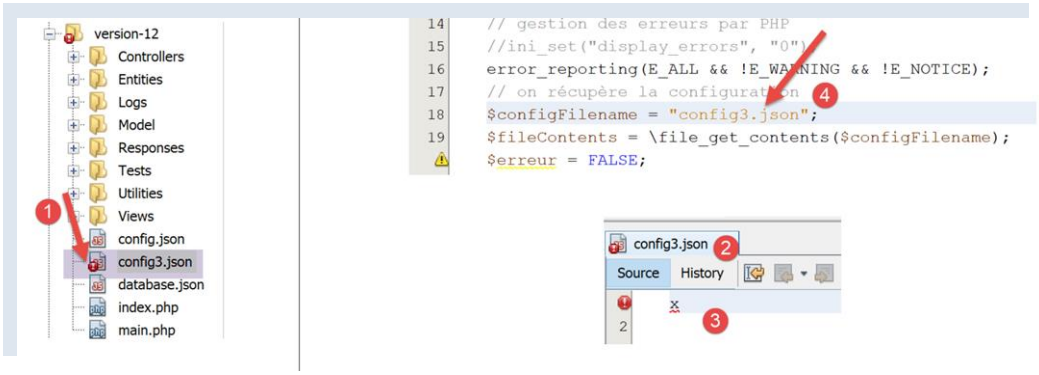


- en **[32]**, le type jSON de la réponse ;

Ce premier test nous a permis de voir qu'on :

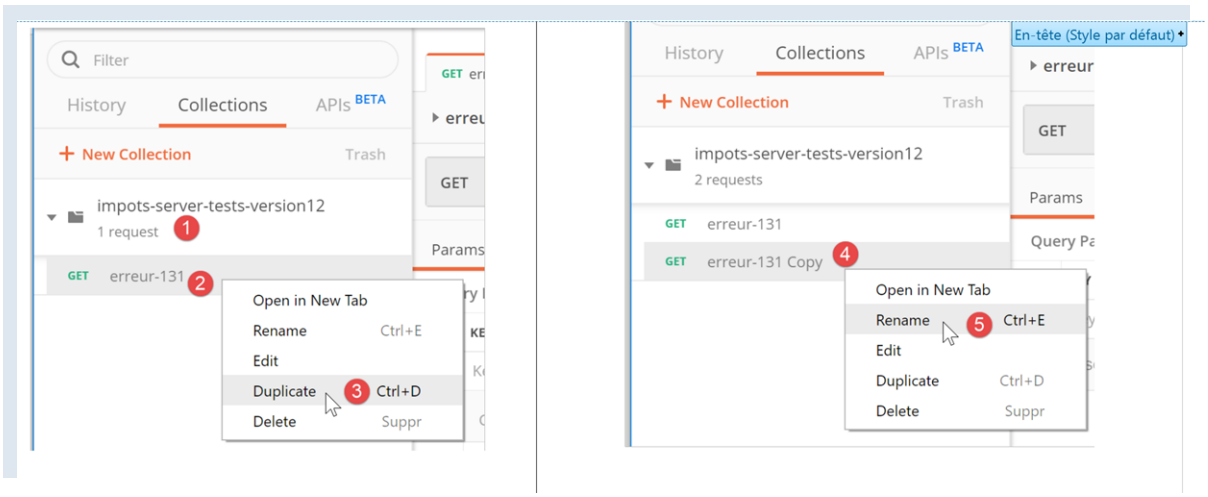
- peut envoyer tout type de requête au serveur testé ;
- peut fixer les paramètres du GET ou du POST ;
- a la totalité de la réponse : entêtes HTTP et le document qui suit ces entêtes **[Body]** ;

Maintenant, faisons un second test :

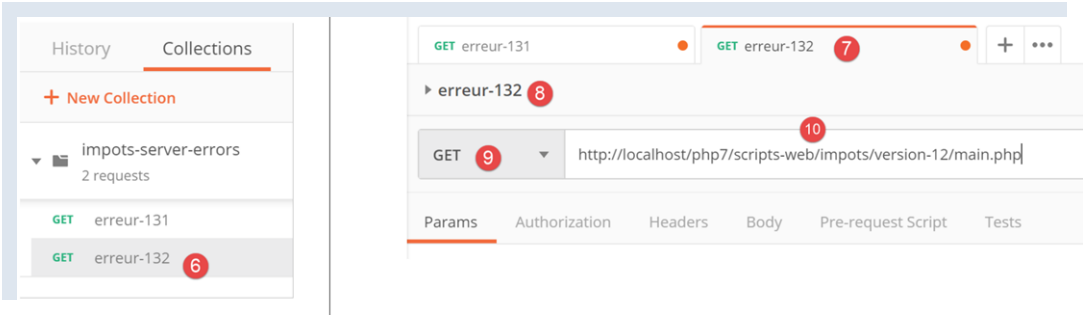


- en **[1-3]**, le fichier **[config3.json]** est un fichier jSON syntaxiquement incorrect ;
- en **[4]**, **[main.php]** est configuré pour utiliser **[config3.json]** ;

Nous ajoutons une nouvelle requête dans **[Postman]** :



- **[1-3]**, on clique droit sur **[2]** et on prend l'option **[duplicate]** pour dupliquer la requête **[2]** ;
- en **[4]**, la nouvelle requête a un nom prédéfini qu'on change en **[5]** ;



- en **[6]**, la requête renommée ;
- en **[9-10]**, on envoie la même requête GET que précédemment ;

- en **[11]**, la réponse jSON du serveur ;

Nous avons montré ici comment allaient être testées les différentes actions du service web du calcul de l'impôt.

1.23.9.4 [main.php] – 2

Nous reprenons l'étude du code du contrôleur principal **[main.php]** :

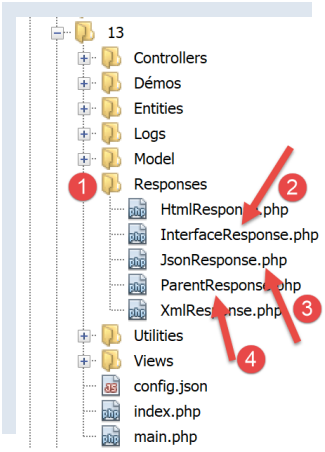
```
1   <?php
2
3   // respect strict des types déclarés des paramètres de foctions
4   declare (strict_types=1);
5
6   // espace de noms
7   namespace Application;
8
9   // dépendances Symfony
10  use Symfony\Component\HttpFoundation\Request;
11  use Symfony\Component\HttpFoundation\Response;
12  use Symfony\Component\HttpFoundation\Session\Session;
13
14  // gestion des erreurs par PHP
15  //ini_set("display_errors", "0");
16  error_reporting(E_ALL && !E_WARNING && !E_NOTICE);
17  // on récupère la configuration
18  $configFilename = "config.json";
19  …
20  // on inclut les dépendances nécessaires au script
21  $rootDirectory = $config["rootDirectory"];
22  foreach ($config["relativeDependencies"] as $dependency) {
23    require_once "$rootDirectory$dependency";
24  }
25  // dépendances absolues (bibliothèques tierces)
26  foreach ($config["absoluteDependencies"] as $dependency) {
27    require_once "$dependency";
28  }
29
30  // création du fichier des logs
31  try {
32    $logger = new Logger($config['logsFilename']);
33  } catch (ExceptionImpots $ex) {
```

```
34    // on n'a pas pu créer le fichier de logs - internal server error
35    $état = 133;
36    (new JsonResponse())->send(
37      NULL, NULL, $config,
38      Response::HTTP_INTERNAL_SERVER_ERROR,
39      ["action" => "non déterminée", "état" => $état, "réponse" => "Le fichier de logs
    [{$config['logsFilename']}] n'a pu être créé"],
40      []);
41    // terminé
42    exit;
43  }
```

Commentaires

- ligne 18 : on a un fichier de configuration **[config.json]** désormais existant et syntaxiquement correct. Il faudrait de plus tester que les clés attendues dans ce fichier sont bien présentes. Nous considèrerons que cela fait partie du travail normal de débogage du développeur. Nous aurions pu faire ce même raisonnement pour les deux précédentes erreurs ;
- lignes 20-28 : on inclut toutes les dépendances nécessaires au projet web. Nous avons déjà rencontré ce code plusieurs fois ;
- ligne 31-43 : on essaie de créer l'objet **[Logger]** qui va nous permettre de loguer des événements dans le fichier **[$config['logsFilename']]**. Cette création peut échouer ;
- lignes 33-43 : gestion de l'erreur de création de l'objet **[Logger]** ;
- ligne 35 : on fixe un n° d'état ;
- lignes 36-40 : on envoie une réponse jSON ;
- ligne 42 : on arrête le script ;

Toutes les réponses envoyées au client implémentent l'interface **[InterfaceResponse]** suivante :



Le code de l'interface **[InterfaceResponse]** est le suivant :

```
1   <?php
2
3   namespace Application;
4
5   // dépendances Symfony
6   use Symfony\Component\HttpFoundation\Request;
7   use Symfony\Component\HttpFoundation\Session\Session;
8
9   interface InterfaceResponse {
10
11    // Request $request : requête en cours de traitement
12    // Session $session : la session de l'application web
13    // array $config : la configuration de l'application
14    // int statusCode : le code HTTP de statut de la réponse
15    // array $content : la réponse du serveur
16    // array $headers : les entêtes HTTP à ajouter à la réponse
17    // Logger $logger : le logueur pour écrire des logs
18
19    public function send(
20      Request $request = NULL,
21      Session $session = NULL,
22      array $config,
23      int $statusCode,
24      array $content,
```

```
25      array $headers,
26      Logger $logger = NULL): void;
27  }
```

- lignes 19-27 : l'interface **[InterfaceResponse]** a une unique méthode **[send]** pour envoyer la réponse au client ;
- lignes 11-17 : la signification des différents paramètres de la méthode **[send]** ;
- lignes 23-25 : les paramètres **[$statusCode, $content, $headers]** sont dans le résultat standard des contrôleurs secondaires de l'application. Cependant la réponse peut avoir besoin d'autres informations. Aussi lui donne-t-on les trois premiers paramètres (lignes 20-22) qui lui donnent accès à la totalité des informations concernant la requête, la session, la configuration ;
- ligne 26 : la réponse a besoin du **[Logger]** car elle va loguer la réponse envoyée au client ;

La classe **[JsonResponse]** implémente l'interface **[InterfaceResponse]** de la façon suivante :

```
1   <?php
2
3   namespace Application;
4
5   // dépendances Symfony
6   use Symfony\Component\Serializer\Encoder\JsonEncode;
7   use Symfony\Component\Serializer\Encoder\JsonEncoder;
8   use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;
9   use Symfony\Component\Serializer\Serializer;
10  use \Symfony\Component\HttpFoundation\Request;
11  use \Symfony\Component\HttpFoundation\Session\Session;
12
13  class JsonResponse extends ParentResponse implements InterfaceResponse {
14
15    // Request $request : requête en cours de traitement
16    // Session $session : la session de l'application web
17    // array $config : la configuration de l'application
18    // int statusCode : le code HTTP de statut de la réponse
19    // array $content : la réponse du serveur
20    // array $headers : les entêtes HTTP à ajouter à la réponse
21    // Logger $logger : le logueur pour écrire des logs
22
23    public function send(
24      Request $request = NULL,
25      Session $session = NULL,
26      array $config,
27      int $statusCode,
28      array $content,
29      array $headers,
30      Logger $logger = NULL): void {
31
32      // préparation sérialiseur symfony
33      $serializer = new Serializer(
34        [
35        // nécessaire pour la sérialisation d'objets
36        new ObjectNormalizer()],
37        // encodeur jSON
38        // pour les options, faire des OU entre les différentes options
39        [new JsonEncoder(new JsonEncode([JsonEncode::OPTIONS => JSON_UNESCAPED_UNICODE]))]
40      );
41      // sérialisation jSON
42      $json = $serializer->serialize($content, 'json');
43      // headers
44      $headers = array_merge($headers, ["content-type" => "application/json"]);
45      // envoi réponse
46      parent::sendResponse($statusCode, $json, $headers);
47      // log
48      if ($logger !== NULL) {
49        $logger->write("réponse=$json\n");
50      }
51    }
52
53  }
```

Commentaires

- ligne 13 : la classe implémente l'interface **[InterfaceResponse]** ;
- ligne 13 : la classe étend la classe **[ParentResponse]**. Tous les types de **[Response]** étendent cette classe. C'est cette classe parent qui envoie la réponse au client (ligne 46). C'est parce que cet code était commun à tous les types de **[Response]** qu'il a été factorisé dans une classe parent ;

- lignes 33-40 : instanciation du sérialiseur **[Symfony]** qui va traduire la réponse du serveur **[$content]** en chaîne jSON (ligne 42) ;
- lignes 34-36 : le 1er paramètre du constructeur de **[Serializer]** est un tableau. Dans celui-ci, on met une instance de la classe **[ObjectNormalizer]** nécessaire à la sérialisation d'objets. Ce cas se présente dans cette application avec une liste de simulations où chaque simulation est une instance de la classe **[Simulation]** ;
- ligne 39 : le second paramètre du constructeur de **[Serializer]** est également un tableau : on y met tous les encodeurs utilisés dans une sérialisation (XML, jSON, CSV…) ;
- ligne 39 : il n'y aura qu'un encodeur ici, de type **[JsonEncoder]**. Le constructeur sans paramètres aurait pu être suffisant. Ici, nous avons passé un paramètre **[JsonEncode]** au constructeur, uniquement pour passer des options d'encodage jSON ;
- ligne 39 : le paramètre du constructeur **[JsonEncode]** est un tableau d'options. Ici on utilise l'option **[JSON_UNESCAPED_UNICODE]** pour demander que les caractères UTF-8 de la chaîne jSON soient rendus nativement et non pas « échappés » ;
- ligne 42 : le corps de la réponse HTTP est sérialisé en jSON grâce au sérialiseur précédent ;
- ligne 44 : on ajoute l'entête HTTP qui dit au client qu'on va lui envoyer du jSON ;
- ligne 46 : on demande à la classe parent d'envoyer la réponse au client ;
- lignes 48-50 : on logue la réponse jSON ;

Le code de la classe parent **[ParentResponse]** est le suivant :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Response;
7.
8.  class ParentResponse {
9.
10.   // int $statusCode : le code HTTP de statut de la réponse
11.   // string $content : le corps de la réponse à envoyer
12.   // selon les cas, c'est une chaîne jSON, XML, HTML
13.   // array $headers : les entêtes HTTP à ajouter à la réponse
14.
15.   public function sendResponse(
16.     int $statusCode,
17.     string $content,
18.     array $headers): void {
19.
20.     // préparation de la réponse texte du serveur
21.     $response = new Response();
22.     $response->setCharset("utf-8");
23.     // code de statut
24.     $response->setStatusCode($statusCode);
25.     // headers
26.     foreach ($headers as $text => $value) {
27.       $response->headers->set($text, $value);
28.     }
29.     // on envoie la réponse
30.     $response->setContent($content);
31.     $response->send();
32.   }
33. }
```

Commentaires

- lignes 10-13 : la signification des trois paramètres de la méthode **[send]** ;
- ligne 17 : on notera que le corps de la réponse est de type **[string]** et donc prêt à être envoyé (ligne 30) ;
- ligne 22 : la réponse aura des caractères UTF-8 ;
- ligne 24 : code de statut HTTP de la réponse ;
- lignes 26-28 : ajout des entêtes HTTP donnés par le code appelant ;
- lignes 30-31 : envoi de la réponse au client ;

Nous avons détaillé tout le cycle d'une réponse jSON. Nous ne reviendrons pas dessus dans la suite. Il faut simplement se rappeler la signature de l'interface **[InterfaceResponse]** :

```
1.  interface InterfaceResponse {
2.
3.    // Request $request : requête en cours de traitement
4.    // Session $session : la session de l'application web
5.    // array $config : la configuration de l'application
6.    // int statusCode : le code HTTP de statut de la réponse
7.    // array $content : la réponse du serveur
```

```
8.     // array $headers : les entêtes HTTP à ajouter à la réponse
9.     // Logger $logger : le logueur pour écrire des logs
10.
11.    public function send(
12.      Request $request = NULL,
13.      Session $session = NULL,
14.      array $config,
15.      int $statusCode,
16.      array $content,
17.      array $headers,
18.      Logger $logger = NULL): void;
19. }
```

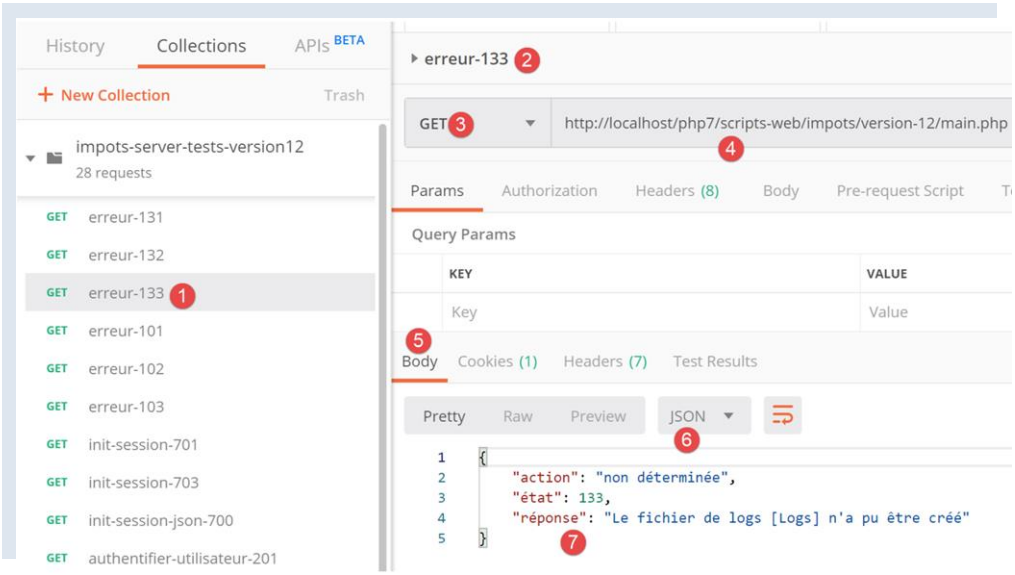
Le contrôleur principal **[main.php]** devra respecter cette signature à chaque fois qu'il demandera l'envoi de la réponse au client.

1.23.9.5 Tests [Postman] – 2

Nous modifions le fichier **[config.json]** de la façon suivante :

- en **[1]**, nous indiquons que le fichier de logs est **[Logs]** qui est un dossier **[2]**. La création du fichier **[Logs]** devrait donc échouer ;

Nous créons une nouvelle requête **[Postman] [3]**, appelée **[erreur-133]** :



- **[2-4]** : nous définissons la même requête que dans les deux tests précédents ;
- **[5-7]** : nous récupérons bien la réponse jSON attendue ;

1.23.9.6 [main.php] – 3

Continuons l'étude du contrôleur principal **[main.php]** :

```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Application;
8.
9.  // dépendances Symfony
10. use Symfony\Component\HttpFoundation\Request;
11. use Symfony\Component\HttpFoundation\Response;
12. use Symfony\Component\HttpFoundation\Session\Session;
13.
14. // gestion des erreurs par PHP
15. …
16.
17. // création du fichier des logs
18. …
19.
20. // 1er log
21. $logger->write("\n---nouvelle requête\n");
22. // requête courante
23. $request = Request::createFromGlobals();
24.
25. // session
26. $session = new Session();
27. $session->start();
28. // liste d'erreurs
29. $erreurs = [];
30. $erreur = FALSE;
31. // on gère l'action demandée
32. if (!$request->query->has("action")) {
33.   $erreurs[] = "paramètre [action] manquant";
34.   $erreur = TRUE;
35.   $état = 101;
36.   $action = "";
37. } else {
38.   // on mémorise l'action
39.   $action = strtolower($request->query->get("action"));
40. }
41. // on logue l'action
42. $logger->write("action [$action] demandée\n");
43.
44. // l'action existe-t-elle ?
45. if (!$erreur && !array_key_exists($action, $config["actions"])) {
46.   $erreurs[] = "action [$action] invalide";
47.   $erreur = TRUE;
48.   $état = 102;
49. }
50.
51. // le type de session doit être connu avant de faire certaines actions
52. if (!$erreur && !$session->has("type") && $action !== "init-session") {
53.   $erreurs[] = "pas de session en cours. Commencer par action [init-session]";
54.   $erreur = TRUE;
55.   $état = 103;
56. }
57.
58. // pour certaines actions on doit être authentifié
59. if (!$erreur && !$session->has("user") && $action !== "authentifier-utilisateur" && $action !== "init-
    session") {
60.   $erreurs[] = "action demandée par utilisateur non authentifié";
61.   $erreur = TRUE;
62.   $état = 104;
63. }
64.
65. // erreurs ?
66. if ($erreurs) {
67.   // on prépare la réponse sans l'envoyer
68.   $statusCode = Response::HTTP_BAD_REQUEST;
69.   $content = ["réponse" => $erreurs];
70.   $headers = [];
71. } else {
72.   // ---------------------------
```

```
73.   // on exécute l'action à l'aide de son contrôleur
74.   $controller = __NAMESPACE__ . $config["actions"][$action];
75.   $logger->write("contrôleur : $controller\n");
76.   list($statusCode, $état, $content, $headers) = (new $controller())->execute($config, $request, $session);
77. }
78.
79. // ---------------------- on envoie la réponse
80. // cas de l'erreur fatale HTTP_INTERNAL_SERVER_ERROR
81. // on envoie un mail à l'administrateur si on peut
82. if ($statusCode === Response::HTTP_INTERNAL_SERVER_ERROR && $config['adminMail'] != NULL) {
83.   $infosMail = $config['adminMail'];
84.   $infosMail['message'] = json_encode($content, JSON_UNESCAPED_UNICODE);
85.   $sendAdminMail = new SendAdminMail($infosMail, $logger);
86.   $sendAdminMail->send();
87. }
88. // la réponse dépend du type de la session
89. if ($session->has("type")) {
90.   // le type de session est dans la session
91.   $type = $session->get("type");
92. } else {
93.   // si pas de type dans session, alors par défaut ce sera une réponse en jSON
94.   $type = "json";
95. }
96. // on ajoute les clés [action, état] à la réponse du contrôleur
97. $content = ["action" => $action, "état" => $état] + $content;
98. // on instancie l'objet [Response] chargée d'envoyer la réponse au client
99. $response = __NAMESPACE__ . $config["types"][$type]["response"];
100.(new $response())->send($request, $session, $config, $statusCode, $content, $headers, $logger);
101.
102.// la réponse a été envoyée - on libère les ressources
103.$logger->close();
104.exit;
```

Commentaires

- une fois que les premières vérifications ont été faites et qu'il sait qu'il peut travailler, le contrôleur principal s'intéresse à l'action qu'on lui a demandée : elle doit remplir certaines conditions ;
- ligne 21 : on logue le fait qu'on a une nouvelle requête. On ne pouvait pas le faire avant car on n'était pas sûr d'avoir un fichier de logs valide ;
- ligne 23 : on encapsule toutes les informations de la requête du client dans l'objet Symfony **[Request]** ;
- ligne 26 : on démarre une nouvelle session où on récupère la session existante si elle existe ;
- ligne 27 : la session est activée ;
- ligne 29 : un tableau de messages d'erreur ;
- ligne 30 : un booléen qui au fil des tests nous dit si on a rencontré ou pas une erreur ;
- ligne 32 : le paramètre **[action]** doit faire partie de l'URL sous la forme **[main.php?action=uneAction]**. Le paramètre **[action]** fait alors partie des paramètres **[$request->query]** ;
- lignes 33-36 : cas de l'absence du paramètre **[action]** dans l'URL. L'erreur est notée et un état **[101]** lui est attribué ;
- ligne 39 : si le paramètre **[action]** est présent dans l'URL, il est mémorisé ;
- ligne 42 : le type de l'action est logué ;
- lignes 45-49 : si le paramètre **[action]** est présent, il doit alors être valide. Toutes les actions autorisées sont définies dans le tableau associatif **[$config["actions"]]** ;
- lignes 46-48 : si l'action est invalide, l'erreur est notée et l'état **[102]** lui est attribué ;
- lignes 52-56 : on a une action valide. Elle doit encore remplir d'autres conditions. L'application web fournit trois types de réponse (jSON, XML, HTML). Ce type est fixé par l'action **[init-session]**. Cette action place le type de la session dans la clé **[type]** ;
- ligne 52 : en-dehors de l'action **[init-session]**, tout autre action doit se dérouler avec une clé **[type]** dans la session ;
- lignes 53-55 : si ce n'est pas le cas, l'erreur est notée et l'état **[103]** lui est attribué ;
- lignes 58-63 : en-dehors des actions **[init-session]** et **[authentifier-utilisateur]**, toutes les autres actions doivent se faire après authentification. Celle-ci se fait à l'aide de l'action **[authentifier-utilisateur]**, qui si l'authentification réussit met une clé **[user]** dans la session ;
- ligne 59 : si l'action n'est ni **[init-session]** ni **[authentifier-utilisateur]** et que la clé **[user]** n'est pas dans la session, alors on a une erreur ;
- lignes 60-62 : on note l'erreur et lui attribue l'état **[104]** ;
- lignes 66-71 : on teste si le tableau **[$erreurs]** est non vide. Si c'est le cas, alors l'action demandée ou son contexte d'exécution sont erronés ;
- lignes 68-70 : on prépare la réponse à envoyer au client mais on ne l'envoie pas encore ;
- ligne 68 : code de statut HTTP ;
- ligne 69 : corps de la réponse ;
- ligne 70 : entêtes à ajouter à la réponse, aucun ici ;
- ligne 73 : on a une action valide. On va demander à son contrôleur (secondaire) de la traiter ;

- ligne 74 : on construit le nom de la classe du contrôleur à exécuter. **[__NAMESPACE__]** est l'espace de noms dans lequel on se trouve, ici **[Application]** (ligne 7) ;
- les noms des classes de contrôleur secondaire sont dans le fichier **[config.json]** :

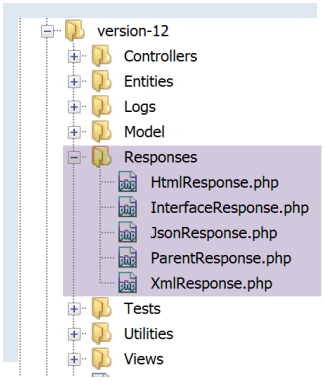
```
1.  "actions":
2.              {
3.                  "init-session": "\\InitSessionController",
4.                  "authentifier-utilisateur": "\\AuthentifierUtilisateurController",
5.                  "calculer-impot": "\\CalculerImpotController",
6.                  "lister-simulations": "\\ListerSimulationsController",
7.                  "supprimer-simulation": "\\SupprimerSimulationController",
8.                  "fin-session": "\\FinSessionController",
9.                  "afficher-calcul-impot": "\\AfficherCalculImpotController"
10. },
```

A chaque action correspond un contrôleur secondaire. Si l'action est **[authentifier-utilisateur]**, la variable **[$controller]** de la ligne 74 aura donc la valeur **[Application/AuthentifierUtilisateurController]** ;

- ligne 75 : on logue le nom du contrôleur secondaire, pour vérification en cours de développement ;
- ligne 76 : le contrôleur secondaire est exécuté. Nous reviendrons sur les contrôleurs secondaires un peu plus loin ;
- ligne 76 : tous les contrôleurs secondaires rendent le même type de résultat qui est un tableau :
  - le 1er élement du tableau **[$statusCode]** est le code de statut HTTP de la réponse à envoyer ;
  - le second élément **[$état]** est l'état de l'application après exécution du contrôleur ;
  - le troisième élément **[$content]** est un tableau associatif avec l'unique clé **[réponse]** qui est le corps de la réponse à envoyer au client ;
  - le quatrième élément **[$headers]** est un tableau d'entêtes HTTP à ajouter à la réponse envoyée au client ;
- ligne 79 : on arrive ici :
  - soit parce qu'il y a eu erreur (lignes 68-70) ;
  - soit après exécution d'un contrôleur (lignes 72-76) ;
  - dans les deux cas, les éléments **[$statusCode, $état, $content, $headers]** nécessaires à l'élaboration de la réponse au client sont connus ;
- lignes 82-87 : traitent le cas particulier du code de statut **[500 Internal Server Error]**. Si un contrôleur a mis ce code de statut, c'est que l'application ne peut pas fonctionner. C'est par exemple le cas du calcul de l'impôt si le SGBD utilisé n'a pas été lancé ou ne répond plus. On envoie alors un mail à l'administrateur de l'application pour l'avertir. Nous ne commenterons pas particulièrement ce code. L'utilisation de la classe **[SendAdminMail]** a déjà été présentée (paragraphe lien) ;
- lignes 89-95 : on détermine le type **[jSON, XML, HTML]** de l'application web. Si l'action **[init-session]** a été exécutée avec succès, ce type est dans la session associé à la clé **[type]** (ligne 91). Si ce n'est pas le cas, alors on fixe arbitrairement un type pour la réponse, le type jSON (ligne 94) ;
- ligne 97 : **[$content]** est un tableau avec une unique clé **[réponse]** et une unique valeur, le corps de la réponse à envoyer au client. On lui ajoute les clés **[action]** et **[état]**. La clé **[action]** permettra de mieux suivre les logs du fichier **[logs.txt]**. La clé **[état]** aura deux rôles :
  - elle permettra aux clients jSON et XML de connaître l'état dans lequel l'action exécutée a mis l'application web ;
  - dans le cas d'une réponse HTML, elle permettra de choisir la vue HTML qu'il faut envoyer au navigateur client ;
- ligne 99 : on choisit le type de classe **[Response]** à exécuter pour envoyer la réponse au client ;

Nous avons déjà présenté la classe **[JsonResponse]** au paragraphe lien. Elle implémente l'interface **[InterfaceResponse]** et étend la classe **[ParentResponse]**. C'est le cas des deux autres classes **[XmlResponse]** et **[HtmlResponse]**.

Les réponses sont rassemblées dans le dossier **[Responses]** :



Toutes ces classes implémentent l'interface **[InterfaceResponse]** présentée également au paragraphe lien :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Request;
7.  use Symfony\Component\HttpFoundation\Session\Session;
8.
9.  interface InterfaceResponse {
10.
11.   // Request $request : requête en cours de traitement
12.   // Session $session : la session de l'application web
13.   // array $config : la configuration de l'application
14.   // int statusCode : le code HTTP de statut de la réponse
15.   // array $content : la réponse du serveur
16.   // array $headers : les entêtes HTTP à ajouter à la réponse
17.   // Logger $logger : le logueur pour écrire des logs
18.
19.   public function send(
20.     Request $request = NULL,
21.     Session $session = NULL,
22.     array $config,
23.     int $statusCode,
24.     array $content,
25.     array $headers,
26.     Logger $logger = NULL): void;
27. }
```

Cette interface a une unique méthode **[send]** chargée d'envoyer la réponse au client. Cette méthode a les 7 paramètres décrits aux lignes 11-17. Toutes les classes et interfaces du dossier **[Responses]** sont dans l'espace de noms **[Application]** (ligne 3).

Revenons au code de **[main.php]** :

```
1.  …
2.  // on ajoute les clés [action, état] à la réponse du contrôleur
3.  $content = ["action" => $action, "état" => $état] + $content;
4.  // on instancie l'objet [Response] chargée d'envoyer la réponse au client
5.  $response = __NAMESPACE__ . $config["types"][$type];
6.  (new $response())->send($request, $session, $config, $statusCode, $content, $headers, $logger);
7.
8.  // la réponse a été envoyée - on libère les ressources
9.  $logger->close();
10. exit;
```

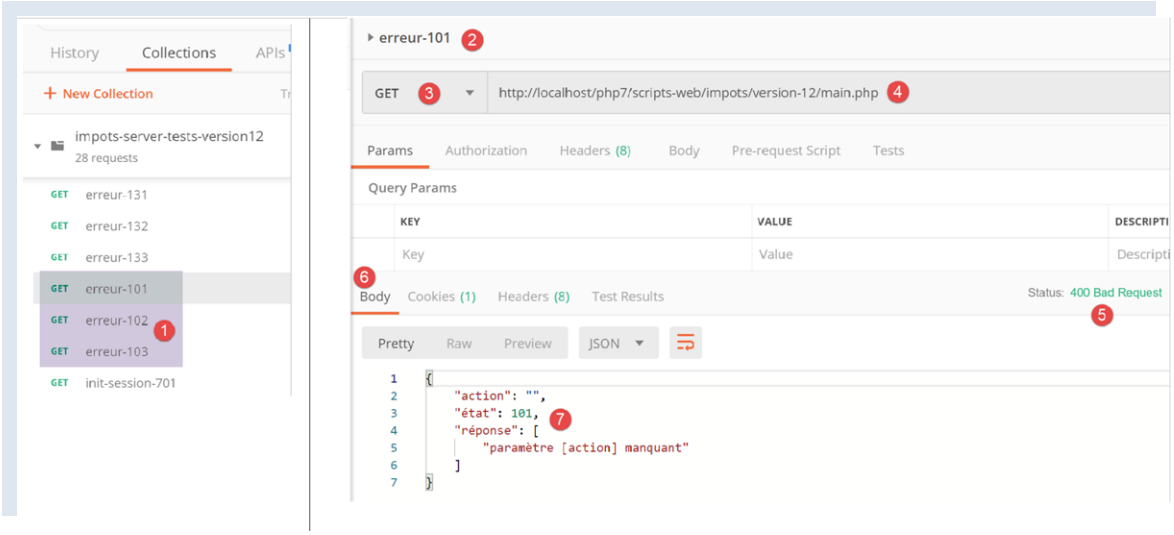
- ligne 5 : on instancie la classe **[Response]** qui convient au type de l'application. Ces classes sont définies dans le fichier **[config.json]** de la façon suivante :

```
"types": {
       "json": "\\JsonResponse",
       "html": "\\HtmlResponse",
       "xml": "\\XmlResponse"
},
```

- ligne 5 : le nom de la classe est préfixée par son espace de noms ;
- ligne 6 : la classe **[Response]** est instanciée et sa méthode **[send]** appelée avec les 7 paramètres qu'elle attend. Ces paramètres sont ceux de l'interface **[InterfaceResponse]** que tous les classes de réponse implémentent. Cela envoie la réponse au client ;
- ligne 9 : on ferme le fichier de logs ;
- ligne 10 : le contrôleur principal a terminé son travail ;

1.23.9.7 Tests [Postman] – 3

On va tester divers cas d'erreur du paramètre **[action]** de l'URL.

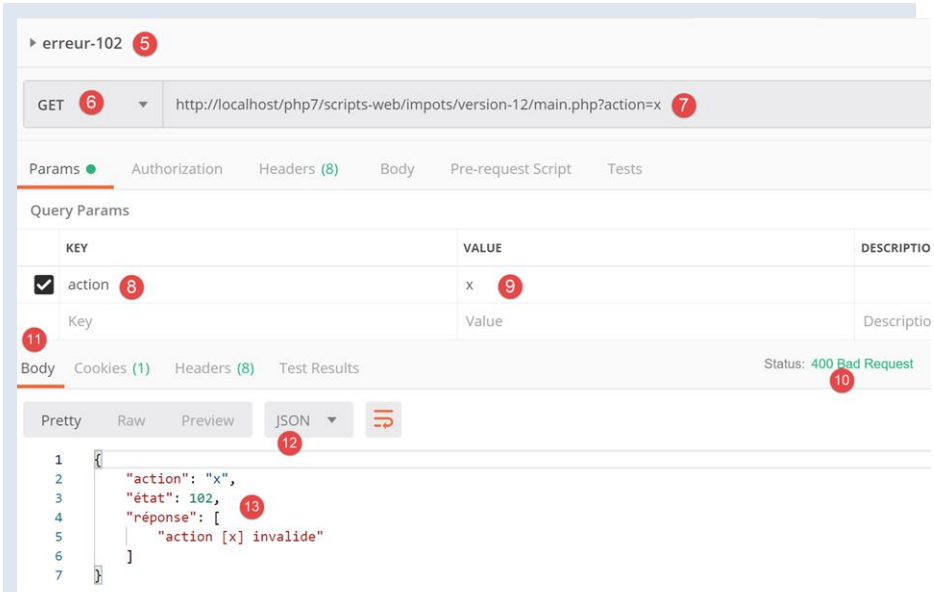


- en **[1]** :
  - **[erreur-101]** : cas du paramètre **[action]** manquant dans l'URL ;
  - **[erreur-102]** : cas du paramètre **[action]** présent dans l'URL mais non reconnu ;
  - **[erreur-103]** : cas du paramètre **[action]** présent dans l'URL, reconnu mais sans que le type de réponse attendue **[json, xml, html]** ait été défini ;

Chaque requête est exécutée. Nous présentons directement les résultats obtenus :

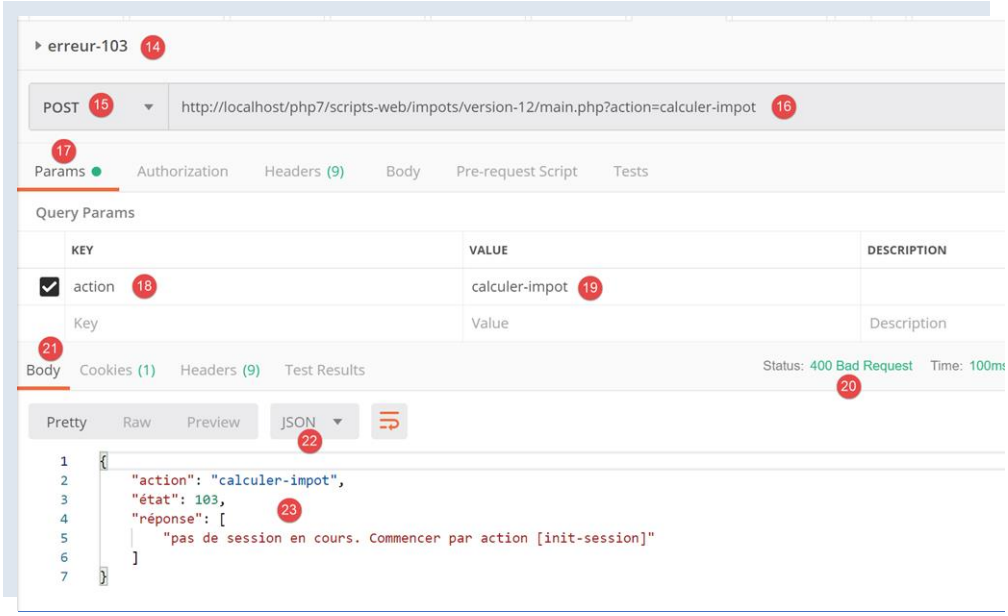
Ci-dessus :

- en **[2-4]**, une requête sans le paramètre **[action]** dans l'URL **[4]** ;
- en **[5-7]**, le résultat jSON ;



Ci-dessus :

- en **[5-9]**, une requête avec un paramètre **[action]** invalide ;
- en **[10-13]**, la réponse jSON ;

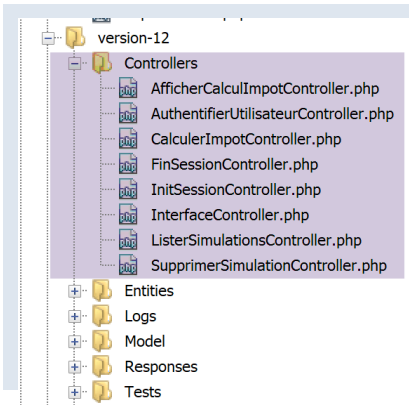


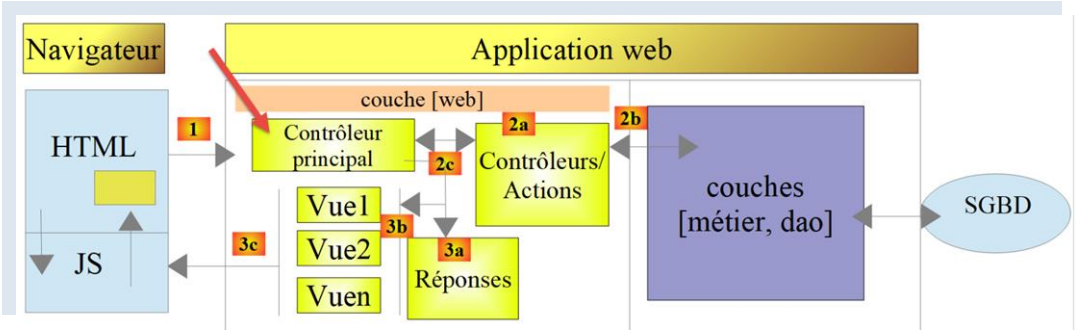
Ci-dessus :

- en **[14-19]**, une action reconnue mais le type (json, xml, html) n'a pas encore été précisé ;
- en **[20-23]**, la réponse jSON du serveur ;

1.23.10 Les contrôleurs secondaires

Chaque action est exécutée par un des contrôleurs du dossier **[Controllers]** :





Dans l'architecture générale de l'application ci-dessus, les contrôleurs secondaires sont en **[2a]**.

Chaque contrôleur implémente l'interface **[InterfaceController]** suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Request;
7.  use Symfony\Component\HttpFoundation\Session\Session;
8.
9.  interface InterfaceController {
10.
11.   // $config est la configuration de l'application
12.   // traitement d'une requête Request
13.   // utile la session Session et peut la modifier
14.   // $infos sont des informations supplémentaires propres à chaque contrôleur
15.
16.   // rend un tableau [$statusCode, $état, $content, $headers]
17.   public function execute(
18.     array $config,
19.     Request $request,
20.     Session $session,
21.     array $infos=NULL): array;
22. }
```

Commentaires

- tous les contrôleurs secondaires sont exécutés via la méthode **[execute]** de la ligne 17. On passe à cette méthode les informations connues du contrôleur principa :
  - ligne 18 : **[array $config]** qui encapsule la configuration de l'application ;
  - ligne 19 : **[Request $request]** qui est la requête HTTP en cours de traitement ;
  - ligne 20 : **[Session $session]** qui est la session courante de l'application web ;
  - ligne 21 : **[array $infos=NULL]** qui est un tableau supplémentaire d'informations pour le contrôleur au cas où les trois premiers paramètres de la méthode ne suffiraient pas. Dans cette application, ce paramètre n'a jamais été utilisé. Il est là par prudence ;
- ligne 21 : la méthode **[execute]** rend le tableau `[$statusCode, $état, $content, $headers]`
  - **[int $statusCode]** : le code de statut de la réponse HTTP ;
  - **[int $état]** : l'état dans lequel se trouve l'application à la fin de l'exécution ;
  - **[array $content]** : un tableau associatif **[réponse=>résultat]** où **[résultat]** est de type quelconque : c'est le résultat produit par le contrôleur et qui sera envoyé au client, une fois ce résultat sérialisé sous la forme d'une chaîne de caractères ;
  - **[array $headers]** : la liste des entêtes HTTP à incorporer à la réponse HTTP du serveur ;

Chaque contrôleur secondaire est appelé par le code suivant du contrôleur principal :

```
1.   // on exécute l'action à l'aide de son contrôleur
2.   $controller = __NAMESPACE__ . $config["actions"][$action];
3.  list($statusCode, $état, $content, $headers) = (new $controller())->execute($config, $request, $session);
```

Ligne 3, on voit que le 4e paramètre **[array $infos=NULL]** de la méthode **[execute]** n'est pas utilisé.

1.23.11 Les actions

Nous passons maintenant en revue les différentes actions possibles du service web :

Action	Rôle	Contexte d'exécution
init-session	Sert à fixer le type (json, xml, html) des réponses souhaitées	Requête **GET main.php?action=init-session&type=x** peut être émise à tout moment
authentifier-utilisateur	Autorise ou non un utilisateur à se connecter	Requête **POST main.php?action=authentifier-utilisateur** La requête doit avoir deux paramètres postés **[user, password]** Ne peut être émise que si le type de la session (json, xml, html) est connu
calculer-impot	Fait une simulation de calcul d'impôt	Requête **POST main.php?action=calculer-impot** La requête doit avoir trois paramètres postés **[marié, enfants, salaire]** Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
lister-simulations	Demande à voir la liste des simulations opérées depuis le début de la session	Requête **GET main.php?action=lister-simulations** La requête n'accepte aucun autre paramètre Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
supprimer-simulation	Supprime une simulation de la liste des simulations	Requête **GET main.php?action=lister-simulations&numéro=x** La requête n'accepte aucun autre paramètre

		Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
fin-session	Termine la session de simulations.	Techniquement l'ancienne session web est supprimée et une nouvelle session est créée Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié

Tous les contrôleurs secondaires procèdent de la même façon :

- ils vérifient leurs paramètres. Ceux-ci sont trouvés dans l'objet **[Request→query]** pour les paramètres présents dans l'URL et dans l'objet **[Request→request]** pour ceux qui sont postés (requête POST) ;
- un contrôleur s'apparente à une foncion ou méthode qui vérifie la validité de ses paramètres. Pour le contrôleur c'est cependant un peu plus compliqué :
  - les paramètres attendus peuvent être absents ;
  - les paramètres attendus sont tous des chaînes de caractères, alors qu'une fonction peut fixer le type de ses paramètres. Si le paramètre attendu est un nombre, alors il faut vérifier que la chaîne du paramètre est bien celui d'un nombre ;
  - une fois vérifié, que les paramètres attendus sont présents et syntaxiquement corrects, il faut vérifier qu'ils sont valides dans le contexte d'exécution du moment. Ce contexte est présent dans la session. L'exemple de l'authentification est un exemple de contexte d'exécution. Certaines actions ne doivent être traitées qu'une fois le client authentifié. Généralement, une clé dans la session indique si cette authentification a eu lieu ou pas ;
  - une fois, les vérifications précédentes faites, le contrôleur secondaire peut travailler. Ce travail de vérification des paramètres est très important. On ne peut pas accepter qu'un client nous envoie n'importe quoi à n'importe quel moment de la vie de l'application. On doit contrôler totalement la vie de celle-ci ;
  - une fois son travail fait, le contrôleur secondaire rend le tableau **[$statusCode, $état, $content, $headers]** attendu par le contrôleur principal qui l'a appelé ;

Nous allons maintenant passer en revue les différents contrôleurs ou ce qui revient au même les différentes actions qui rythment la vie de l'application web.

1.23.11.1 L'action [init-session]

L'action **[init-session]** est traitée par le contrôleur **[InitSessionController]** suivant :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Request;
7.  use Symfony\Component\HttpFoundation\Response;
8.  use Symfony\Component\HttpFoundation\Session\Session;
9.
10. class InitSessionController implements InterfaceController {
11.
12.   // $config est la configuration de l'application
13.   // traitement d'une requête Request
14.   // utile la session Session et peut la modifier
15.   // $infos sont des informations supplémentaires propres à chaque contrôleur
16.
17.   // rend un tableau [$statusCode, $état, $content, $headers]
18.   public function execute(
19.     array $config,
20.     Request $request,
21.     Session $session,
22.     array $infos = NULL): array {
23.
24.     // on doit avoir un GET et un unique paramètre autre que [action]
25.     $method = strtolower($request->getMethod());
26.     $erreur = $method !== "get" || $request->query->count() != 2;
27.     if ($erreur) {
28.       $état = 701;
29.       $message = "méthode GET exigée avec paramètres [action, type] dans l'URL";
30.       return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
31.     }
32.     // on récupère le paramètres du GET
33.     $erreur = FALSE;
34.     // type
35.     if (!$request->query->has("type")) {
36.       $erreur = TRUE;
37.       $état = 702;
38.       $message = "paramètre [type] manquant";
```

```
39.     } else {
40.       $type = strtolower($request->query->get("type"));
41.     }
42.     // vérification du type
43.     if (!$erreur && !array_key_exists($type, $config["types"])) {
44.       $erreur = TRUE;
45.       $état = 703;
46.       $message = "paramètre type [$type] invalide";
47.     }
48.     // erreur ?
49.     if ($erreur) {
50.       return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
51.     }
52.     // on met le type de session dans la session
53.     $session->set("type", $type);
54.     // message de réussite
55.     $message = "session démarrée avec type [$type]";
56.     $état = 700;
57.     return [Response::HTTP_OK, $état, ["réponse" => $message], []];
58.   }
59.
60. }
```

Commentaires

- on attend une requête **[GET main.php?action=init-session&type=xxx]**
- lignes 25-26 : on vérifie que la requête est une requête GET avec deux paramètres dans l'URL ;
- lignes 27-31 : si ce n'est pas le cas, on note l'erreur et on envoie un résultat **[$statusCode, $état, $content, $headers]** au contrôleur principal ;
- lignes 35-39 : on vérifie que le paramètre **[type]** est bien présent dans l'URL. Si ce n'est pas le cas, on note l'erreur ;
- ligne 40 : on note le type de la session ;
- lignes 43-47 : on vérifie que le type de la session est l'un des termes (json, xml, html). Si ce n'est pas le cas, on note l'erreur ;
- lignes 49-51 : s'il y a eu erreur, on envoie un résultat **[$statusCode, $état, $content, $headers]** au contrôleur principal ;
- ligne 53 : le type de la session est mis dans la session de l'application web ;
- lignes 55-57 : le contrôleur a fini son travail. On envoie un résultat **[$statusCode, $état, $content, $headers]** de succès au contrôleur principal ;

Rappelons ce que fait le contrôleur principal de la réponse des contrôleurs secondaires :

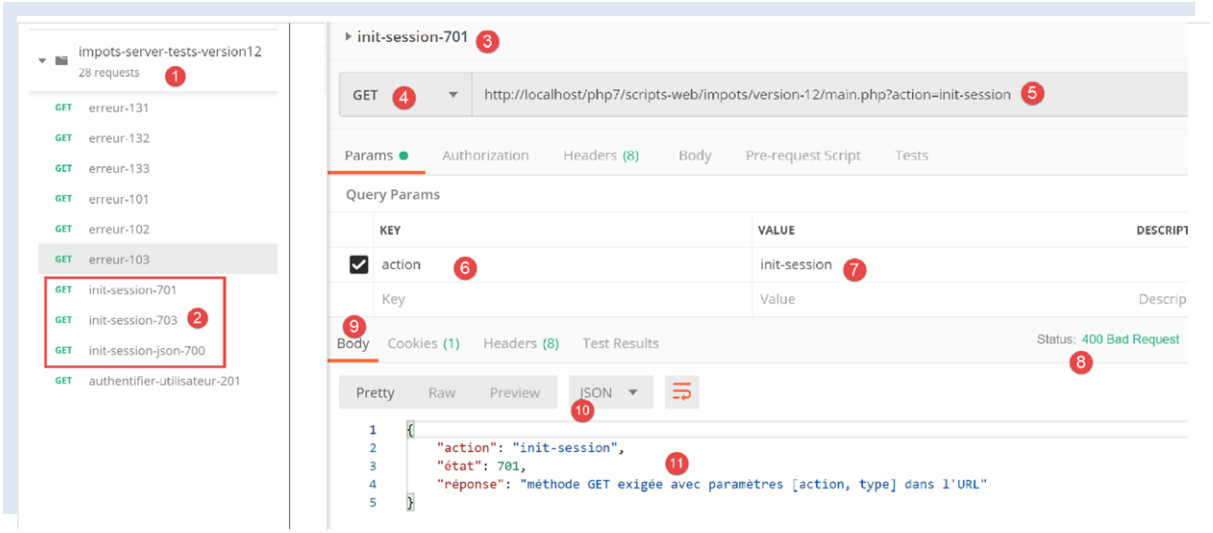
```
1.  // erreurs ?
2.  if ($erreurs) {
3.    // on prépare la réponse sans l'envoyer
4.    $statusCode = Response::HTTP_BAD_REQUEST;
5.    $content = ["réponse" => $erreurs];
6.    $headers = [];
7.  } else {
8.    // --------------------------
9.    // on exécute l'action à l'aide de son contrôleur
10.   $controller = __NAMESPACE__ . $config["actions"][$action];
11.   $logger->write("contrôleur : $controller\n");
12.   list($statusCode, $état, $content, $headers) = (new $controller())->execute($config, $request, $session);
13. }
14.
15. // --------------------- on envoie la réponse
16. // cas de l'erreur fatale HTTP_INTERNAL_SERVER_ERROR
17. // on envoie un mail à l'administrateur si on peut
18. if ($statusCode === Response::HTTP_INTERNAL_SERVER_ERROR && $config['adminMail'] != NULL) {
19.   $infosMail = $config['adminMail'];
20.   $infosMail['message'] = json_encode($content, JSON_UNESCAPED_UNICODE);
21.   $sendAdminMail = new SendAdminMail($infosMail, $logger);
22.   $sendAdminMail->send();
23. }
24. // la réponse dépend du type de la session
25. if ($session->has("type")) {
26.   // le type de session est dans la session
27.   $type = $session->get("type");
28. } else {
29.   // si pas de type dans session, alors par défaut ce sera une réponse en jSON
30.   $type = "json";
31. }
32. // on ajoute les clés [action, état] à la réponse du contrôleur
33. $content = ["action" => $action, "état" => $état] + $content;
34. // on instancie l'objet [Response] chargée d'envoyer la réponse au client
35. $response = __NAMESPACE__ . $config["types"][$type]["response"];
```

```
36. (new $response())->send($request, $session, $config, $statusCode, $content, $headers, $logger);
37.
38. // la réponse a été envoyée - on libère les ressources
39. $logger->close();
40. exit;
```

- ligne 12 : le contrôleur principal récupère le résultat du contrôleur secondaire ;
- lignes 35-36 : après quelques vérifications, il envoie la réponse en instanciant l'une des classes **[JsonResponse, XmlResponse, HtmlResponse]** selon le type (json, xml, html) de la session en cours ;

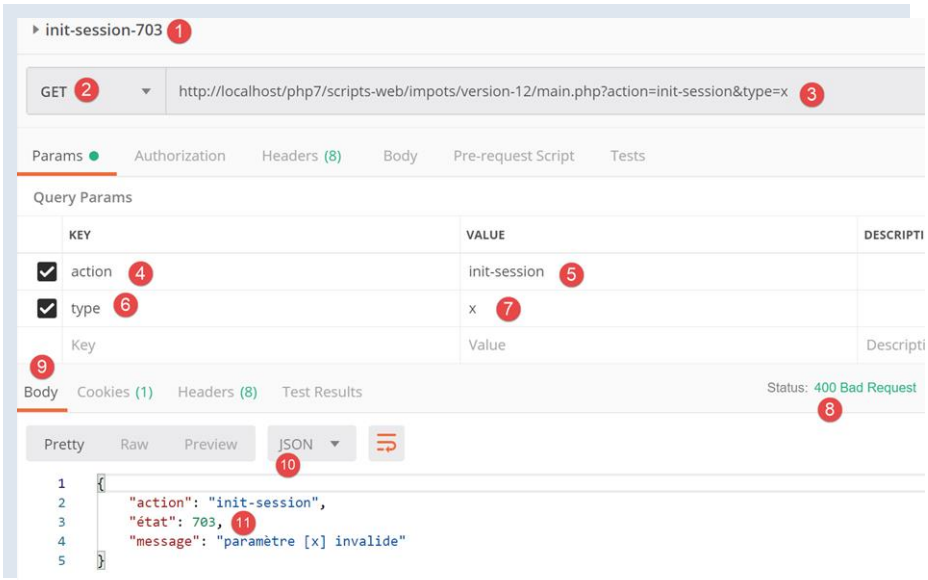
Dans la suite, nous ferons des tests **[Postman]** dans le cadre d'une session de simulations avec le type **[json]**. Le fonctionnement de la classe **[JsonResponse]** a été présenté au paragraphe lien.

1.23.11.2 Tests [Postman]



Ci-dessus :

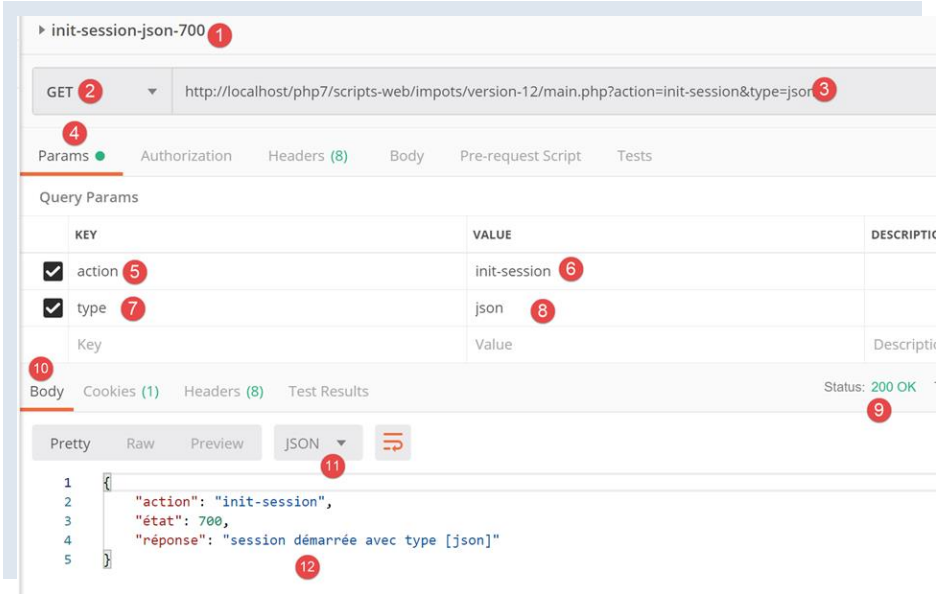
- en **[2]**, trois nouveaux tests ;
- en **[3-7]**, l'action **[init-session]** avec le paramètre **[type]** manquant ;
- en **[8-11]**, la réponse jSON du serveur ;



Ci-dessus :

- en **[1-7]**, l'action **[init-session]** avec un paramètre **[type]** incorrect ;

- en **[8-11]**, la réponse jSON du serveur ;



Ci-dessus :

- en **[1-8]**, l'action **[init-session]** avec le type jSON ;
- en **[9-12]**, la réponse jSON du serveur ;

1.23.11.3 L'action [authentifier-utilisateur]

L'action **[authentifier-utilisateur]** est exécutée par le contrôleur **[AuthentifierUtilisateurController]** suivant :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use \Symfony\Component\HttpFoundation\Response;
7.  use \Symfony\Component\HttpFoundation\Request;
8.  use \Symfony\Component\HttpFoundation\Session\Session;
9.
10. class AuthentifierUtilisateurController implements InterfaceController {
11.
12.   // $config est la configuration de l'application
13.   // traitement d'une requête Request
14.   // utile la session Session et peut la modifier
15.   // $infos sont des informations supplémentaires propres à chaque contrôleur
16.   // rend un tableau [$statusCode, $état, $content, $headers]
17.   public function execute(
18.     array $config,
19.     Request $request,
20.     Session $session,
21.     array $infos = NULL): array {
22.
23.     // on doit avoir un POST et un unique paramètre GET
24.     $method = strtolower($request->getMethod());
25.     $erreur = $method !== "post" || $request->query->count() != 1;
26.     if ($erreur) {
27.       $état = 201;
28.       $message = "méthode POST requise, paramètre [action] dans l'URL, paramètres postés [user,password]";
29.       // on rend le résultat au contrôleur principal
30.       return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
31.     }
32.     // on récupère les paramètres du POST
33.     $erreurs = [];
34.     // user
35.     $état = 210;
36.     if (!$request->request->has("user")) {
37.       $état += 2;
38.       $erreurs[] = "paramètre [user] manquant";
```

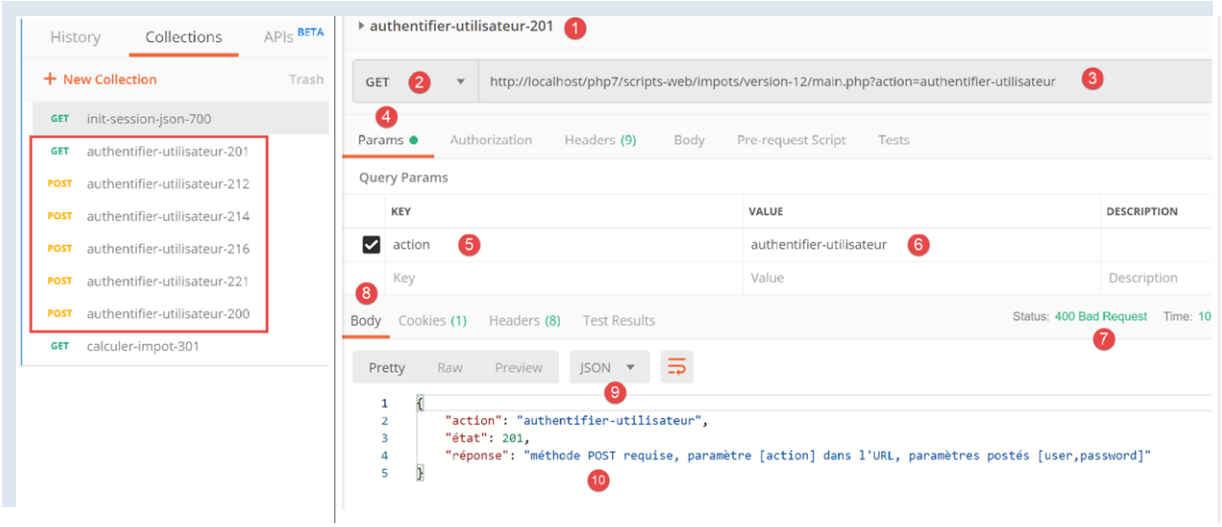
```
39.     } else {
40.       $user = $request->request->get("user");
41.     }
42.     // password
43.     if (!$request->request->has("password")) {
44.       $état += 4;
45.       $erreurs[] = "paramètre [password] manquant";
46.     } else {
47.       $password = trim($request->request->get("password"));
48.     }
49.     // erreur ?
50.     if ($erreurs) {
51.       // on rend le résultat au contrôleur principal
52.       return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $erreurs], []];
53.     }
54.     // vérification des identifiants de l'utilisateur
55.     // l'utilisateur existe-t-il ?
56.     $users = $config["users"];
57.     $i = 0;
58.     $trouvé = FALSE;
59.     while (!$trouvé && $i < count($users)) {
60.       $trouvé = ($user === $users[$i]["login"] && $users[$i]["passwd"] === $password);
61.       $i++;
62.     }
63.     // trouvé ?
64.     if (!$trouvé) {
65.       // message d'erreur
66.       $message = "Echec de l'authentification [$user, $password]";
67.       $état = 221;
68.       // on rend le résultat au contrôleur principal
69.       return [Response::HTTP_UNAUTHORIZED, $état, ["réponse" => $message], []];
70.     } else {
71.       // on note dans la session qu'on a authentifié l'utilisateur
72.       $session->set("user", TRUE);
73.       // message de réussite
74.       $message = "Authentification réussie [$user, $password]";
75.       $état = 200;
76.       // on rend le résultat au contrôleur principal
77.       return [Response::HTTP_OK, $état, ["réponse" => $message], []];
78.     }
79.   }
80.
81. }
```

Commentaires

- on attend une requête **[POST main.php?action=authentifier-utilisateur]** avec deux paramètres postés **[user, password]** ;
- lignes 24-25 : on vérifie qu'on a une requête POST avec un unique paramètre dans l'URL ;
- lignes 26-31 : si erreur il y a, on la note et on rend un résultat **[$statusCode, $état, $content, $headers]** au contrôleur principal ;
- lignes 36-39 : on vérifie la présence du paramètre **[user]** dans les valeurs postées. S'il n'est pas présent, on note l'erreur ;
- lignes 43-45 : on vérifie la présence du paramètre **[password]** dans les valeurs postées. S'il n'est pas présent, on note l'erreur ;
- lignes 50-53 : si l'une des valeurs postées est manquante, un résultat **[$statusCode, $état, $content, $headers]** est rendu au contrôleur principal ;
- lignes 56-62 : on vérifie que le couple **[$user,$password]** récupéré est présent dans le tableau **[$config['users']]** du fichier de configuration ;
- lignes 64-69 : si ce n'est pas le cas, l'erreur est notée. Le code de statut HTTP est mis à **[Response::HTTP_UNAUTHORIZED]** et le résultat **[$statusCode, $état, $content, $headers]** rendu au contrôleur principal ;
- ligne 72 : l'authentification a réussi. On le note dans la session en plaçant dans celle-ci la clé **[user]**. C'est la présence de cette clé qui indique une authentification réussie ;
- lignes 73-77 : on rend un résultat **[$statusCode, $état, $content, $headers]** de réussite au contrôleur principal ;

1.23.11.4 Tests [Postman]

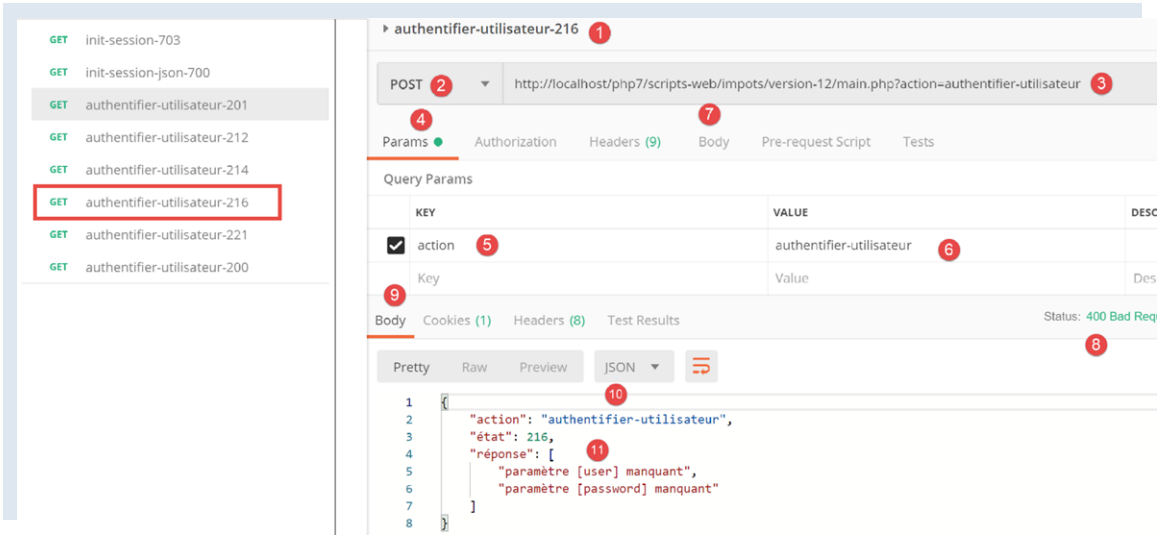
Nous procédons aux tests **[Postman]** du contrôleur **[AuthentifierUtilisateurController]** en mode jSON ;



Ci-dessus :

- en **[1-6]**, l'action **[authentifier-utilisateur]** avec un GET **[2]**, alors qu'il faut un POST ;
- en **[7-10]**, la réponse jSON du serveur ;

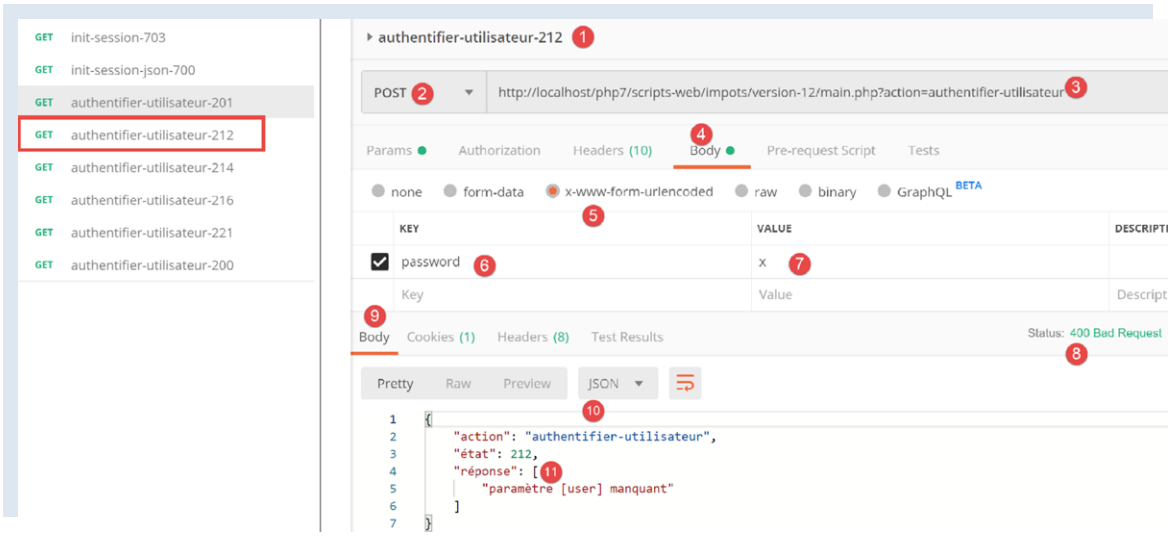
Remplaçons le GET par un POST **[2]** sans mettre de paramètres dans le corps de la réponse **[7]** :



Ci-dessus :

- en **[1-7]**, le POST sans paramètres postés en **[7]** ;
- en **[8-11]**, la réponse jSON du serveur ;

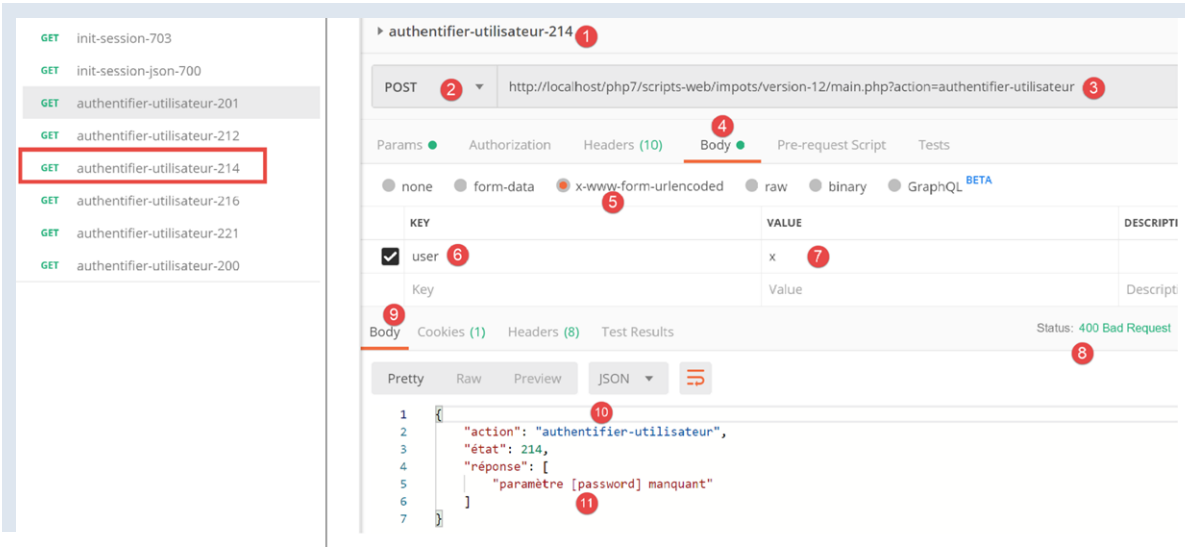
Ajoutons maintenant un paramètre **[password]** dans le corps (body) **[4]** de la requête :



Ci-dessus :

- en **[1-6]**, un requête POST **[2]** avec un paramètre **[password]** posté **[4-6]**. Les paramètres postés doivent être ajoutés dans le corps (body) de la requête **[4]**. Il y a plusieurs façons de poster des valeurs au serveur. Nous choisissons la méthode **[x-www-form-urlencoded] [5]** ;
- en **[8-10]**, la réponse jSON du serveur ;

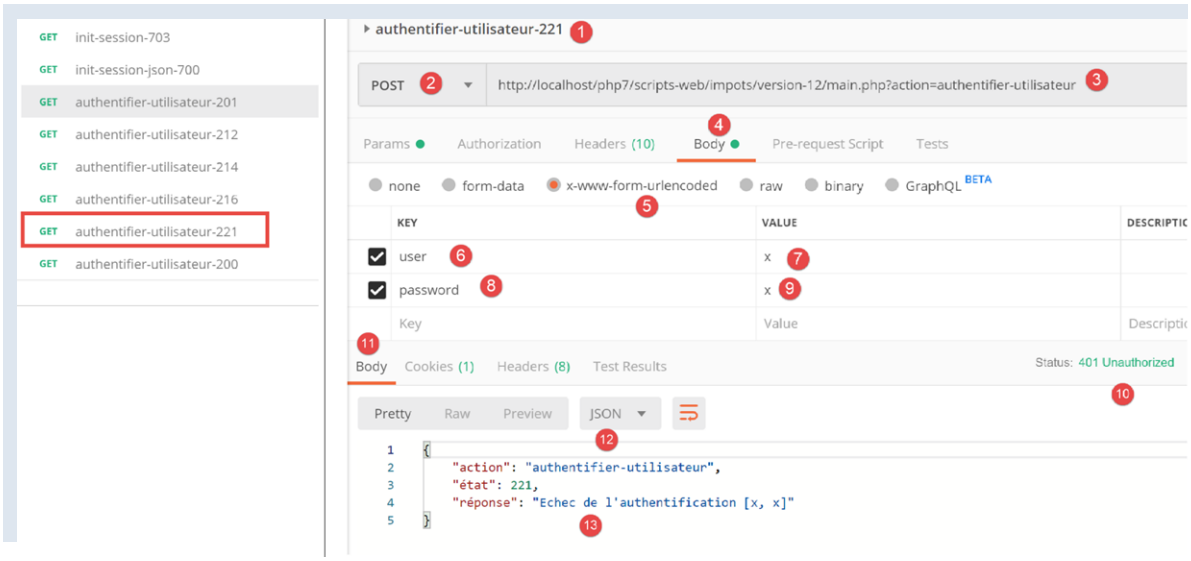
Maintenant définissons le paramètre **[user]** sans le paramètre **[password]** :



Ci-dessus :

- en **[1-7]**, une requête POST sans le paramètre **[password] [4-7]** ;
- en **[8-11]**, la réponse jSON du serveur ;

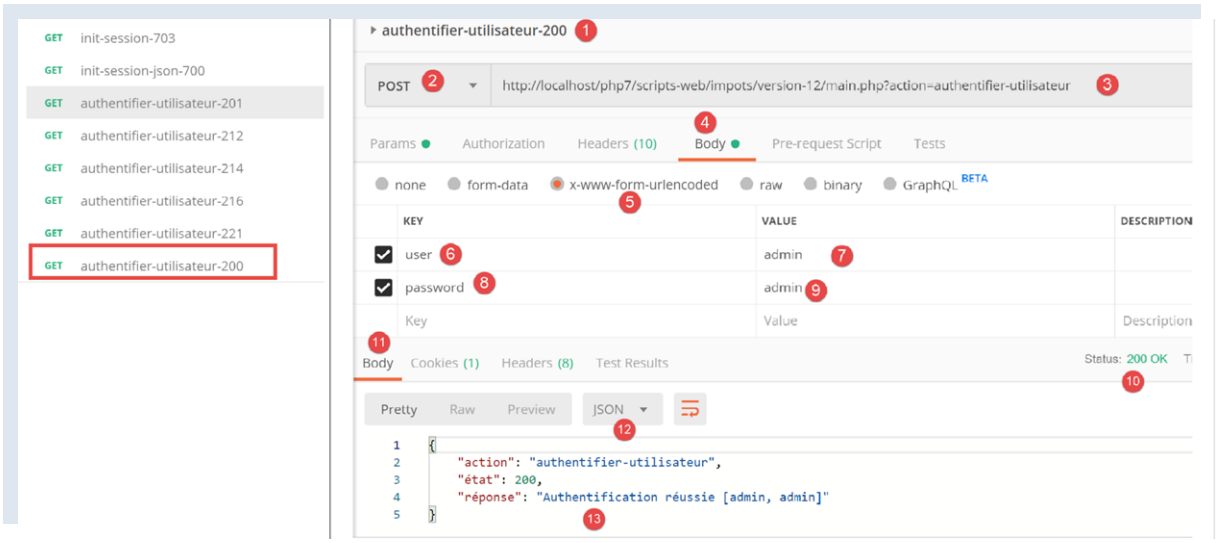
Maintenant définissons les deux paramètres postés **[user, password]** mais avec des valeurs qui font que l'authentification échoue :



Ci-dessus :

- en **[1-9]**, une requête POST avec des paramètres postés **[user, password]** incorrects ;
- en **[10-13]**, la réponse jSON du serveur. On remarquera le code de statut **[401 Unauthorized] [10]** de la réponse ;

Maintenant une requête POST avec des identifiants valides :



Ci-dessus :

- en **[1-9]**, la requête POST **[2]** avec des identifiants valides **[6-9]** ;
- en **[10-13]**, la réponse jSON du serveur. On remarquera le code de statut HTTP **[200 OK]** en **[10]** ;

1.23.11.5 L'action [calculer-impot]

L'action **[calculer-impot]** est traitée par le contrôleur **[CalculerImpotController]** suivant :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use \Symfony\Component\HttpFoundation\Response;
7.  use \Symfony\Component\HttpFoundation\Request;
8.  use \Symfony\Component\HttpFoundation\Session\Session;
9.  // alias de la couche [dao]
```

```php
use \Application\ServerDaoWithSession as ServerDaoWithRedis;

class CalculerImpotController implements InterfaceController {

  // $config est la configuration de l'application
  // traitement d'une requête Request
  // utile la session Session et peut la modifier
  // $infos sont des informations supplémentaires propres à chaque contrôleur
  // rend un tableau [$statusCode, $état, $content, $headers]
  public function execute(
    array $config,
    Request $request,
    Session $session,
    array $infos = NULL): array {

    // on doit avoir un paramètre GET et trois paramètres POST
    $method = strtolower($request->getMethod());
    $erreur = $method !== "post" || $request->query->count() != 1;
    if ($erreur) {
      // on note l'erreur
      $message = "il faut utiliser la méthode [post] avec [action] dans l'URL et les paramètres postés [marié, enfants, salaire]";
      $état = 301;
      // retour résultat au contrôleur principal
      return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
    }
    // on récupère les paramètres du POST
    $erreurs = [];
    $état = 310;
    // statut marital
    if (!$request->request->has("marié")) {
      $état += 2;
      $erreurs[] = "paramètre [marié] manquant";
    } else {
      $marié = trim(strtolower($request->request->get("marié")));
      $erreur = $marié !== "oui" && $marié !== "non";
      if ($erreur) {
        $état += 4;
        $erreurs[] = "valeur [$marié] invalide pour le paramètre [marié]";
      }
    }
    // on récupère le nombre d'enfants
    if (!$request->request->has("enfants")) {
      $état += 8;
      $erreurs[] = "paramètre [enfants] manquant";
    } else {
      $enfants = trim($request->request->get("enfants"));
      $erreur = !preg_match("/^\d+$/", $enfants);
      if ($erreur) {
        $état += 9;
        $erreurs[] = "valeur [$enfants] invalide pour le paramètre [enfants]";
      }
    }
    // on récupère le salaire annuel
    if (!$request->request->has("salaire")) {
      $erreurs[] = "paramètre [salaire] manquant";
      $état += 16;
    } else {
      $salaire = trim($request->request->get("salaire"));
      $erreur = !preg_match("/^\d+$/", $salaire);
      if ($erreur) {
        $état += 17;
        $erreurs[] = "valeur [$salaire] invalide pour le paramètre [salaire]";
      }
    }
    // erreur ?
    if ($erreurs) {
      // retour résultat au contrôleur principal
      return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $erreurs], []];
    }

    // on a tout ce qu'il faut pour travailler
    // Redis
    \Predis\Autoloader::register();
    try {
      // client [predis]
      $redis = new \Predis\Client();
```

```
86.       // on se connecte au serveur pour voir s'il est là
87.       $redis->connect();
88.     } catch (\Predis\Connection\ConnectionException $ex) {
89.       // ça s'est mal passé
90.       // retour résultat avec erreur au contrôleur principal
91.       $état = 350;
92.       return [Response::HTTP_INTERNAL_SERVER_ERROR, $état,
93.         ["réponse" => "[redis], " . utf8_encode($ex->getMessage())], []];
94.     }
95.
96.     // on a des paramètres valides
97.     // création de la couche [dao]
98.     if (!$redis->get("taxAdminData")) {
99.       try {
100.        // on va chercher les données fiscales en base
101.        $dao = new ServerDaoWithRedis($config["databaseFilename"], NULL);
102.        // on met dans redis les données récupérées
103.        $redis->set("taxAdminData", $dao->getTaxAdminData());
104.      } catch (\RuntimeException $ex) {
105.        // ça s'est mal passé
106.        // retour résultat avec erreur au contrôleur principal
107.        $état = 340;
108.        return [Response::HTTP_INTERNAL_SERVER_ERROR, $état,
109.          ["réponse" => utf8_encode($ex->getMessage())], []];
110.      }
111.    } else {
112.      // les données fiscales sont prises dans la mémoire de portée [application]
113.      $arrayOfAttributes = \json_decode($redis->get("taxAdminData"), true);
114.      $taxAdminData = (new TaxAdminData())->setFromArrayOfAttributes($arrayOfAttributes);
115.      // isntanciation de la couche [dao]
116.      $dao = new ServerDaoWithRedis(NULL, $taxAdminData);
117.    }
118.    // création de la couche [métier]
119.    $métier = new ServerMetier($dao);
120.
121.    // on a tout ce qu'il faut pour travailler - calcul de l'impôt
122.    $résultat = $métier->calculerImpot($marié, (int) $enfants, (int) $salaire);
123.    // on ajoute dans la session la simulation qui vient d'être faite
124.    $simulation = new Simulation();
125.    $résultat = ["marié" => $marié, "enfants" => $enfants, "salaire" => $salaire] + $résultat;
126.    $simulation->setFromArrayOfAttributes($résultat);
127.    // existe-t-il une liste de simulations en session ?
128.    if (!$session->has("simulations")) {
129.      $simulations = [];
130.    } else {
131.      $simulations = $session->get("simulations");
132.    }
133.    // ajout de la simulation à la liste de simulations
134.    $simulations[] = $simulation;
135.    // on remet les simulations en session
136.    $session->set("simulations", $simulations);
137.    // retour résultat au contrôleur principal
138.    $état = 300;
139.    return [Response::HTTP_OK, $état, ["réponse" => $résultat], []];
140.  }
141.
142.}
```

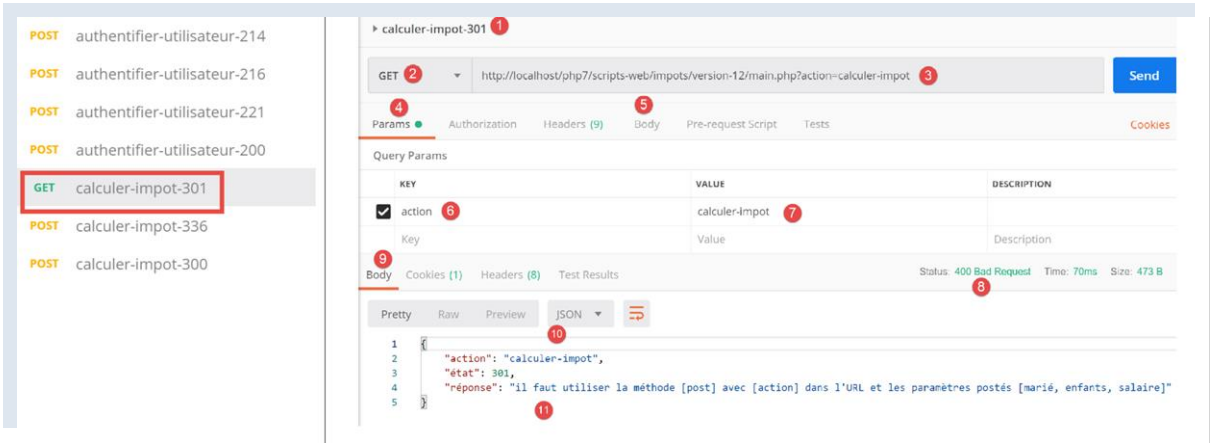
Commentaires

- la requête attendue est **[POST main.php?action=calculer-impot]** avec trois paramètres postés **[marié, enfants, salaire]** :
  - **[marié]** doit avoir sa valeur dans **[oui, non]** ;
  - **[enfants, salaire]** doivent être des entiers positifs ou nuls ;
- lignes 26-27 : on vérifie qu'on a bien un POST avec un unique paramètre dans l'URL ;
- lignes 28-34 : si ce n'est pas le cas, un résultat d'erreur est envoyé au contrôleur principal ;
- ligne 36 : on va cumuler les messages d'erreur dans le tableau **[$erreurs]** ;
- lignes 39-41 : on vérifie la présence du paramètre **[marié]**. S'il n'est pas présent, l'erreur est notée ;
- lignes 43-49 : on vérifie que **[marié]** a sa valeur dans **[oui, non]**. Si ce n'est pas le cas, l'erreur est notée ;
- lignes 51-54 : on vérifie la présence du paramètre **[enfants]**. S'il n'est pas présent, l'erreur est notée ;
- lignes 55-61 : on vérifie que la valeur du paramètre **[enfants]** est un nombre positif ou nul. Si ce n'est pas le cas, l'erreur est notée ;
- lignes 63-66 : on vérifie la présence du paramètre **[salaire]**. S'il n'est pas présent, l'erreur est notée ;
- lignes 67-72 : on vérifie que la valeur du paramètre **[salaire]** est un nombre positif ou nul. Si ce n'est pas le cas, l'erreur est notée ;

- lignes 75-78 : si le tableau **[$erreurs]** n'est pas vide, c'est qu'il y a eu des erreurs. On met le tableau des erreurs dans la réponse et on rend le résultat au contrôleur principal ;
- ligne 80 : on a des paramètres valides. On peut calculer l'impôt. Il faut pour cela construire les couches **[dao]** et **[métier]** qui savent faire ce calcul ;
- lignes 82-94 : on crée un client **[Redis]** ;
- lignes 88-94 : si on n'a pas pu se connecter au serveur **[Redis]**, on envoie un code **[500 Internal Server Error]** au client ;
- ligne 98 : on regarde si le serveur **[Redis]** a la clé **[taxAdminData]**. Cette clé représente les données de l'administration fiscale. Si la clé n'est pas présente, alors les données fiscales doivent être cherchées dans la base de données ;
- ligne 101 : construction de la couche **[dao]** lorsque les données fiscales doivent être prises en base. La classe **[ServerDaoWithRedis]** a été décrite au paragraphe lien ;
- ligne 103 : les données récupérées en base sont mises en mémoire **[Redis]** avec la clé **[taxAdminData]** ;
- lignes 104-110 : si la recherche en base s'est mal passée, on note l'erreur renvoyée par la couche **[dao]** et on l'intègre au résultat renvoyé au contrôleur principal ;
- ligne 109 : le message d'erreur renvoyé par la couche **[PDO]** est codé en **[iso-8859-1]**. On le code en **[utf-8]** ;
- lignes 111-117 : si la clé **[taxAdminData]** existe dans la mémoire **[Redis]**, alors les données fiscales sont passées directement au constructeur de la couche **[dao]** ;
- ligne 119 : la couche **[métier]** est créée. La classe **[ServerMetier]** a été décrite au paragraphe lien ;
- lignes 124-126 : avec le montant de l'impôt calculé, un objet **[Simulation]** est créé. La classe **[Simulation]** encapsule les données d'une simulation et a été décrite au paragraphe lien ;
- lignes 128-132 : la simulation qui vient d'être construite doit être ajoutée à la liste des simulations déjà calculées. Cette liste se trouve en session sauf si aucune simulation n'a encore été faite ;
- ligne 133-136 : la simulation est ajoutée à la liste des simulations et celle-ci est remise en session ;
- lignes 137-139 : on rend le résultat au contrôleur principal ;

1.23.11.6 Tests [Postman]

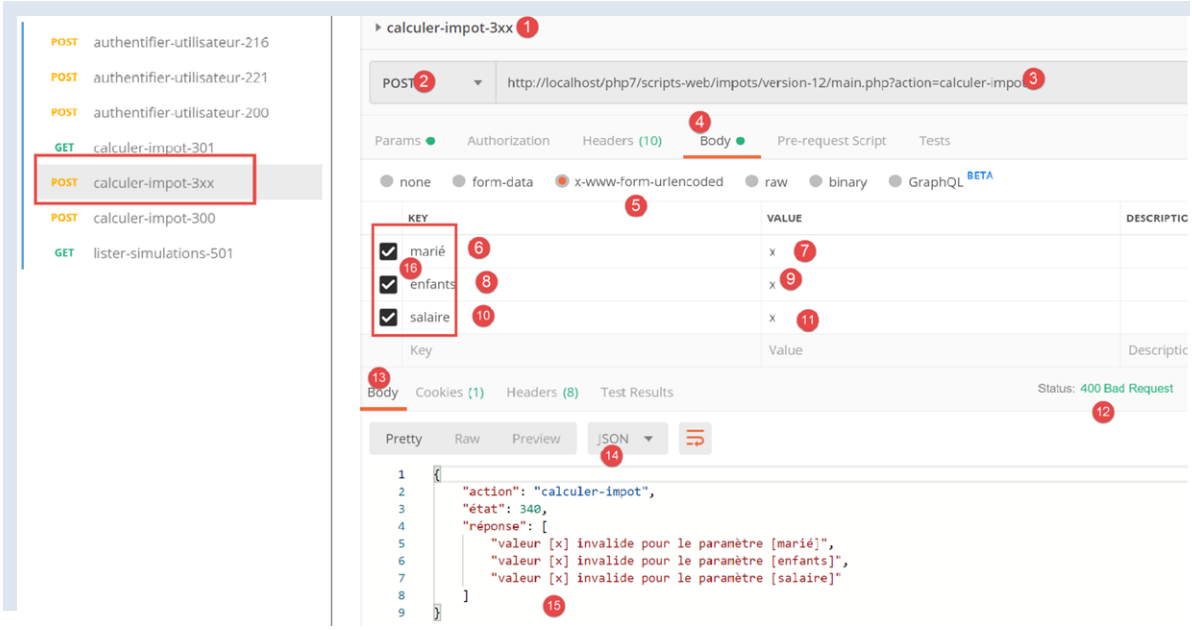
Nous procédons aux tests **[Postman]** du contrôleur **[CalculerImpotController]** en mode jSON ;



Ci-dessus :

- en **[1-7]** on fait une requête **[GET]** au lieu de **[POST]** ;
- en **[8-11]**, la réponse jSON du serveur ;

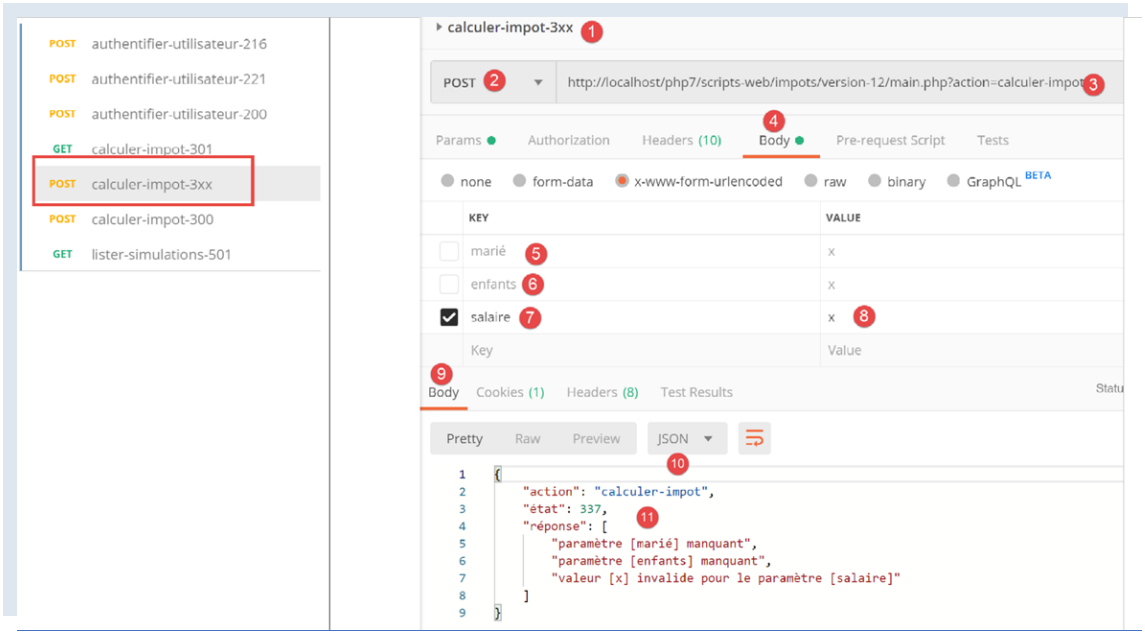
Maintenant, utilisons une méthode **[POST]**, avec ou sans paramètres postés ainsi qu'avec des paramètres postés invalides :



Ci-dessus :

- on fait une requête **[POST] [2]** avec des paramètres postés **[6-11] [marié, enfants, salaire]** invalides. On peut ne pas poster l'un de ces paramètres en décochant sa case dans **[16]**. Cela vous permettra de tester différents cas de figure. Sur la copie d'écran ci-dessus, les trois paramètres sont présents et tous invalides ;
- en **[12-15]**, la réponse jSON du serveur ;

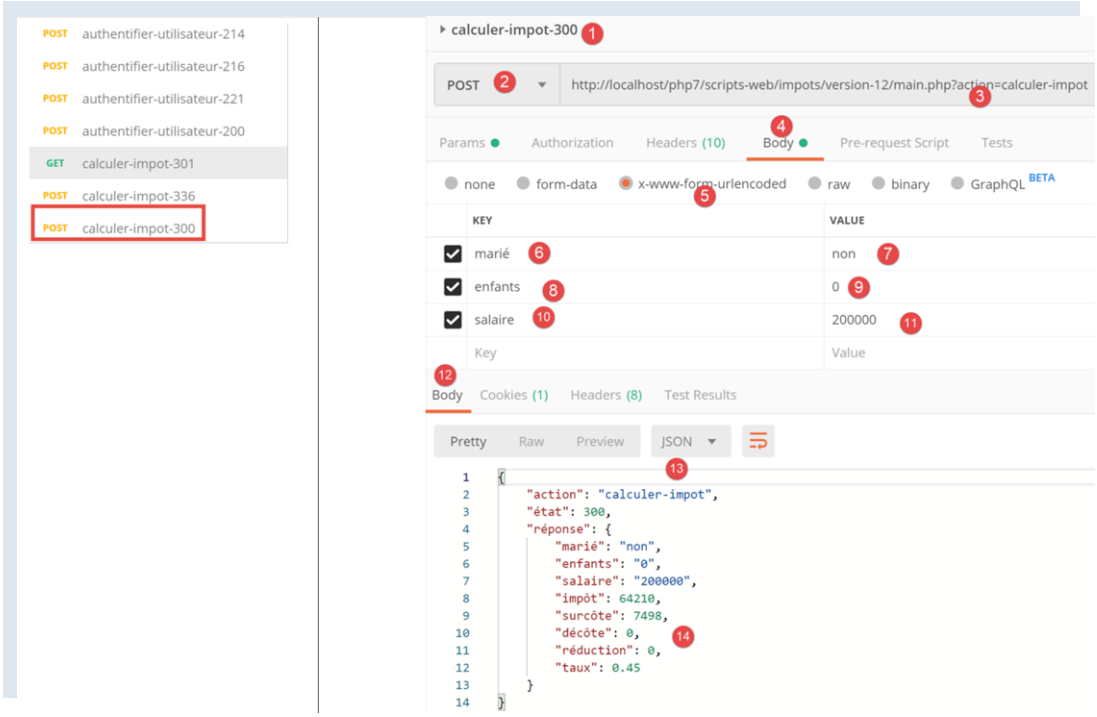
Maintenant décochons deux des trois paramètres postés :



Ci-dessus,

- en **[5-8]**, seul le paramètre **[salaire]** est posté et de plus il est invalide ;
- en **[9-11]**, le résultat jSON du serveur ;

Maintenant faisons un calcul d'impôt avec des paramètres valides :



Ci-dessus :

- en **[1118]**, une demande avec des paramètres valides **[6-8]** ;
- en **[12-14]**, la réponse jSON du serveur ;

1.23.11.7 L'action [lister-simulations]

L'action **[lister-simulations]** est traitée par le contrôleur secondaire **[ListerSimulationsController]** suivant :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use \Symfony\Component\HttpFoundation\Response;
7.  use \Symfony\Component\HttpFoundation\Request;
8.  use \Symfony\Component\HttpFoundation\Session\Session;
9.
10. class ListerSimulationsController {
11.
12.   // $config est la configuration de l'application
13.   // traitement d'une requête Request
14.   // utile la session Session et peut la modifier
15.   // $infos sont des informations supplémentaires propres à chaque contrôleur
16.   // rend un tableau [$statusCode, $état, $content, $headers]
17.   public function execute(
18.     array $config,
19.     Request $request,
20.     Session $session,
21.     array $infos = NULL): array {
22.
23.     // on doit avoir un unique paramètre GET
24.     $method = strtolower($request->getMethod());
25.     $erreur = $method !== "get" || $request->query->count() != 1;
26.     if ($erreur) {
27.       $état = 501;
28.       $message = "GET requis, avec l'unique paramètre [action] dans l'URL";
29.       // on rend un résultat avec erreur au contrôleur principal
30.       return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
31.     }
32.     // on récupère la liste des simulations dans la session
33.     if (!$session->has("simulations")) {
34.       $simulations = [];
35.     } else {
36.       $simulations = $session->get("simulations");
```

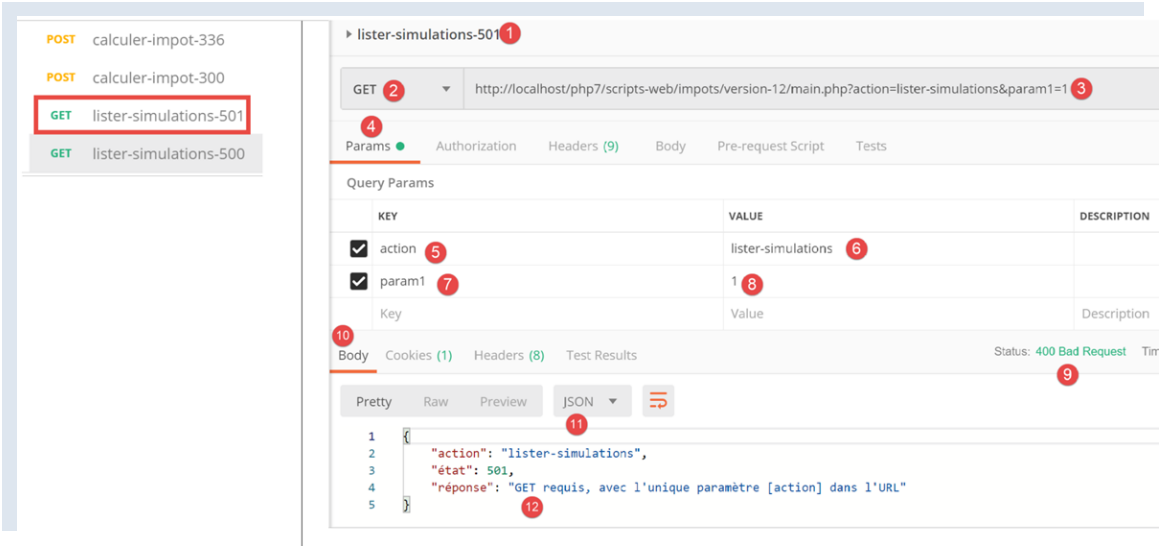
```
37.     }
38.     // on rend un résultat avec sucès au contrôleur principal
39.     $état = 500;
40.     return [Response::HTTP_OK, $état, ["réponse" => $simulations], []];
41.   }
42.
43. }
```

Commentaires

- requête **[GET main.php?action=lister-simulations]** ;
- lignes 24-25 : on vérifie qu'on a une requête GET avec un unique paramètre ;
- lignes 26-31 : si ce n'est pas le cas, un résultat avec erreur est rendu au contrôleur principal ;
- lignes 33-37 : on récupère la liste des simulations dans la session si elle s'y trouve (ligne 36), sinon cette liste est vide (ligne 34) ;
- lignes 39-40 : on rend la liste des simulations au contrôleur principal ;

1.23.11.8 Tests [Postman]

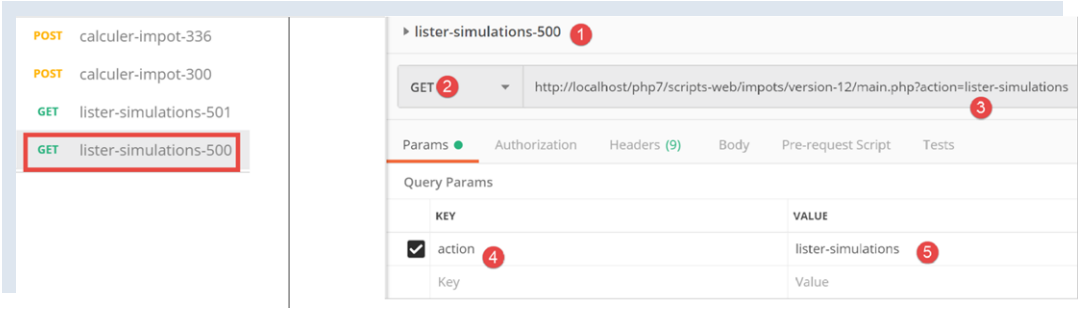
Nous allons créer deux tests, un d'erreur et un réussi.



Ci-dessus :

- en **[1-8]**, on fait une requête **[GET]** avec un paramètre **[param1]** en trop dans l'URL **[3, 7-8]** ;
- en **[9-12]**, la réponse jSON du serveur ;

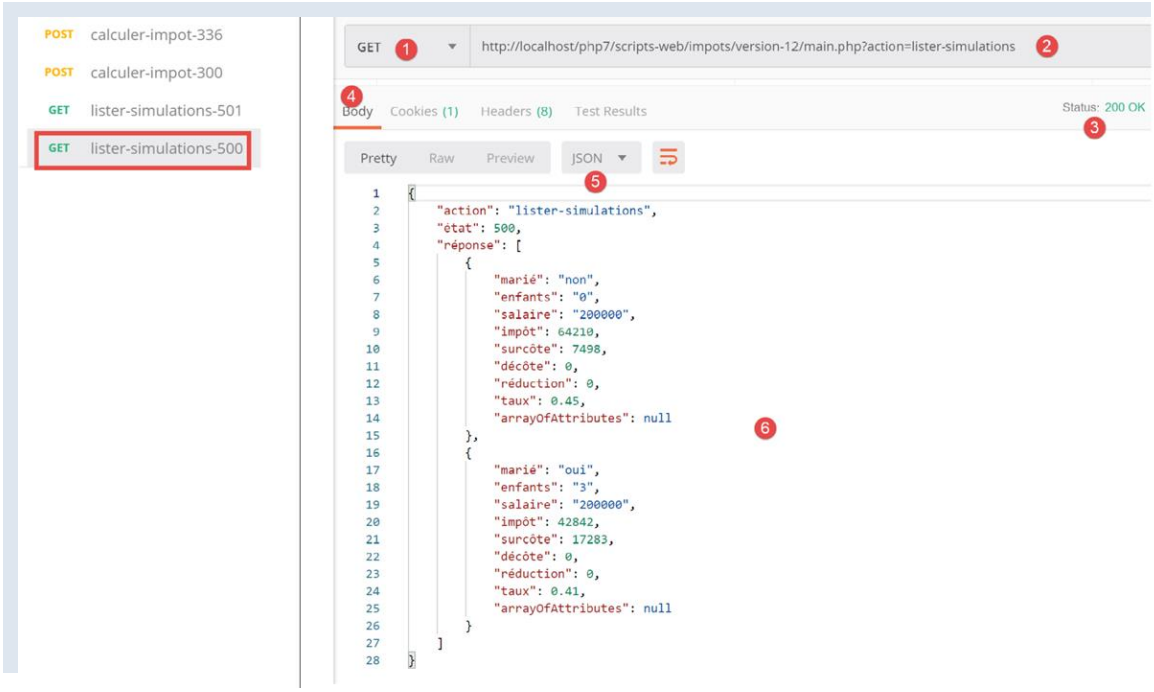
Maintenant faisons une requête valide :



Ci-dessus :

- en **[1-5]**, une requête valide ;

Le résultat de la requête est le suivant :



- en **[3-6]**, la réponse jSON du serveur. Avant ce test, le test **[Postman] [calculer-impot-300]** avait été exécuté plusieurs fois pour créer des simulations dans la session web du serveur ;

1.23.11.9 L'action [supprimer-simulation]

L'action **[supprimer-simulation]** est traitée par le contrôleur secondaire **[SupprimerSessionController]** suivant :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use \Symfony\Component\HttpFoundation\Response;
7.  use \Symfony\Component\HttpFoundation\Request;
8.  use \Symfony\Component\HttpFoundation\Session\Session;
9.
10. class SupprimerSimulationController {
11.
12.   /// $config est la configuration de l'application
13.   // traitement d'une requête Request
14.   // utile la session Session et peut la modifier
15.   // $infos sont des informations supplémentaires propres à chaque contrôleur
16.   // rend un tableau [$statusCode, $état, $content, $headers]
17.   public function execute(
18.     array $config,
19.     Request $request,
20.     Session $session,
21.     array $infos = NULL): array {
22.
23.     // on doit avoir deux paramètres GET
24.     $method = strtolower($request->getMethod());
25.     $erreur = $method !== "get" || $request->query->count() != 2;
26.     $état = 600;
27.     if ($erreur) {
28.       $état += 2;
29.       $message = "GET requis, avec les paramètres [action, numéro]";
30.     }
31.     // le paramètre [numéro] doit exister
32.     if (!$erreur) {
33.       $état += 4;
34.       $erreur = !$request->query->has("numéro");
35.       if ($erreur) {
36.         $message = "paramètre [numéro] manquant";
37.       }
38.     }
```

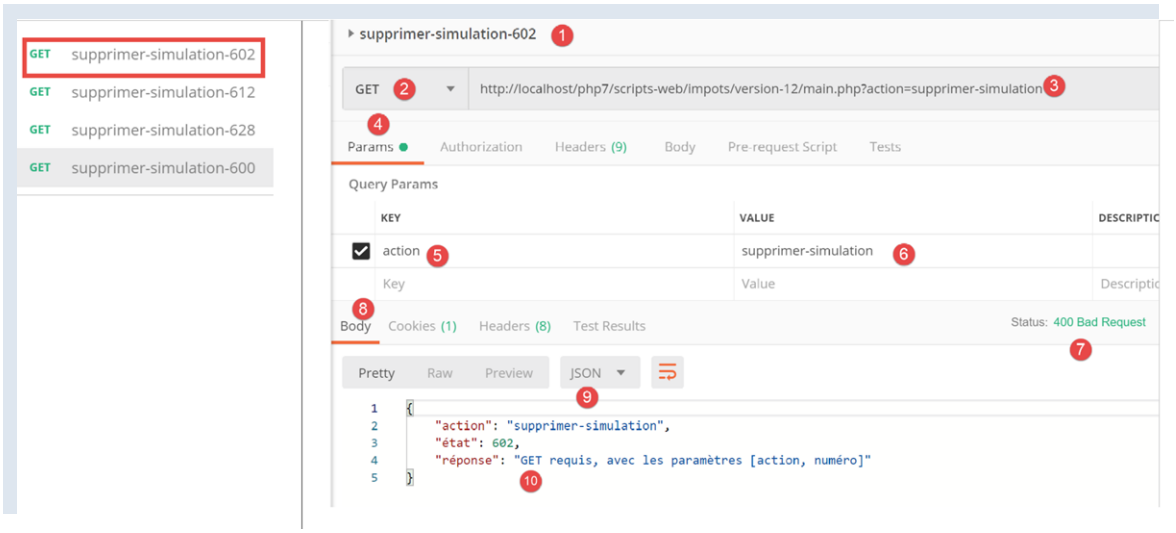
```
39.     // le paramètre [numéro] doit être valide
40.     if (!$erreur) {
41.       $état += 8;
42.       $numéro = $request->query->get("numéro");
43.       $erreur = !preg_match("/^\d+$/", $numéro);
44.       if ($erreur) {
45.         $message = "paramètre [$numéro] invalide";
46.       }
47.     }
48.     // le paramètre [numéro] doit être dans l'intervalle [0,n-1]
49.     // si n est le nombre de simulations
50.     if (!$erreur) {
51.       $numéro = (int) $numéro;
52.       $erreur = !$session->has("simulations");
53.       if (!$erreur) {
54.         $simulations = $session->get("simulations");
55.         $erreur = $numéro < 0 || $numéro >= count($simulations);
56.       }
57.       if ($erreur) {
58.         $état += 16;
59.         $message = "la simulation n° [$numéro] n'existe pas";
60.       }
61.     }
62.     // erreur ?
63.     if ($erreur) {
64.       // on rend le résultat au contrôleur principal
65.       return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
66.     }
67.     // on supprime la simulation $numéro
68.     unset($simulations[$numéro]);
69.     $simulations = array_values($simulations);
70.     // on remet les simulations dans la session
71.     $session->set("simulations", $simulations);
72.     // on rend la liste des simulations au client
73.     $état = 600;
74.     return [Response::HTTP_OK, $état, ["réponse" => $simulations], []];
75.   }
76.
77. }
```

Commentaires

- requête **[GET main.php?action=supprimer-simulation&numéro=x]** ;
- lignes 24-30 : on vérifie qu'on a une requête GET avec deux paramètres ;
- lignes 32-38 : on vérifie que le paramètre **[numéro]** existe dans les paramètres de l'URL ;
- lignes 40-47 : on vérifie que la valeur du paramètre **[numéro]** est syntaxiquement correcte ;
- lignes 50-61 : on vérifie que la simulation n° **[numéro]** existe bien. Il y a deux cas d'erreur :
  - la liste de simulations ne peut être trouvée dans la session (ligne 52) ;
  - le n° **[numéro]** de la simulation à supprimer n'existe pas dans la liste des simulations ;
- lignes 63-66 : en cas d'erreur, un résultat avec erreur est rendu au contrôleur principal ;
- ligne 68 : la simulation n° **[numéro]** est supprimée ;
- ligne 69 : l'opération **[unset]** ne change pas les index **[0, n-1]** de la liste. Pour les mettre à jour, on demande les valeurs du tableau **[$simulations]** pour éliminer la simulation manquante ;
- ligne 71 : on remet le nouveau tableau des simulations dans la session ;
- lignes 73-74 : on rend au contrôleur principal la nouvelle liste des simulations ;

1.23.11.10 Tests [Postman]

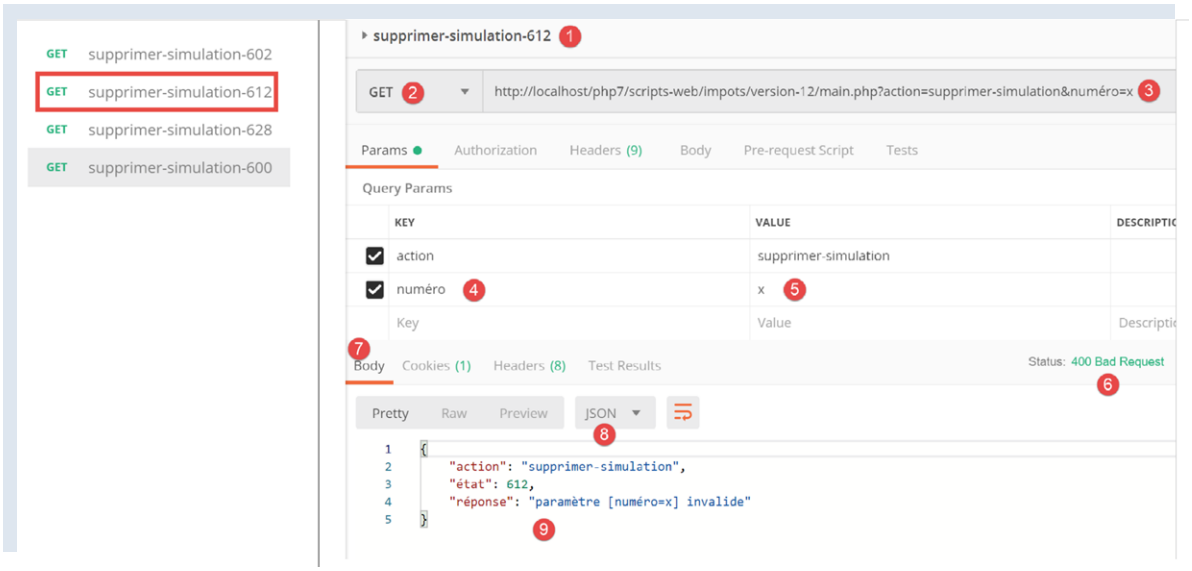
Nous allons faire des tests d'erreur et de succès :



Ci-dessus :

- en **[1-6]**, une requête GET sans le paramètre **[numéro]** ;
- en **[7-10]**, la réponse jSON du serveur ;

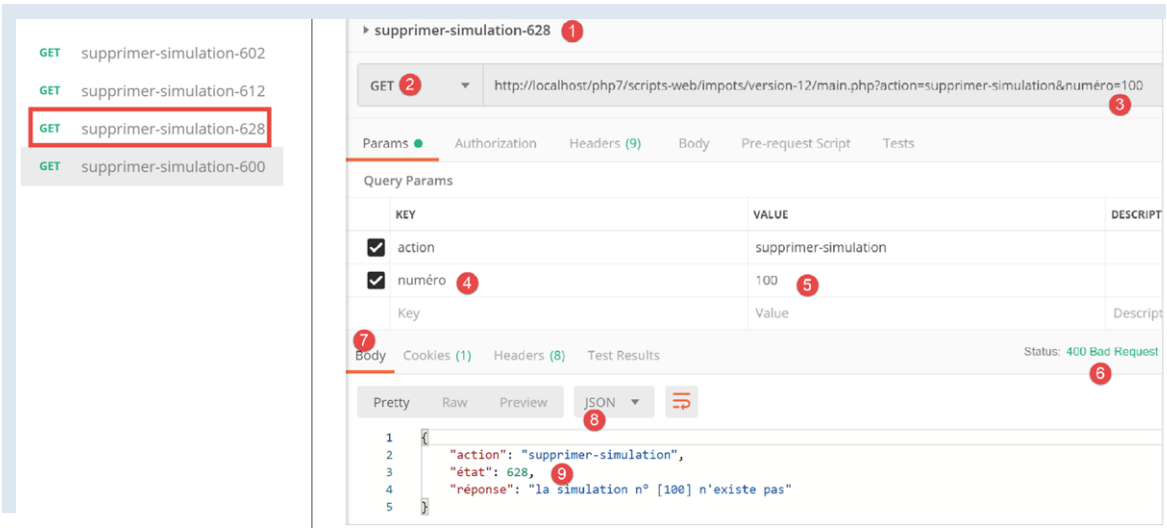
Maintenant une requête avec un n° syntaxiquement incorrect :



Ci-dessus :

- en **[1-5]**, une requête GET avec un paramètre **[numéro]** invalide **[3, 5]** ;
- en **[6-9]**, la réponse jSON du serveur ;

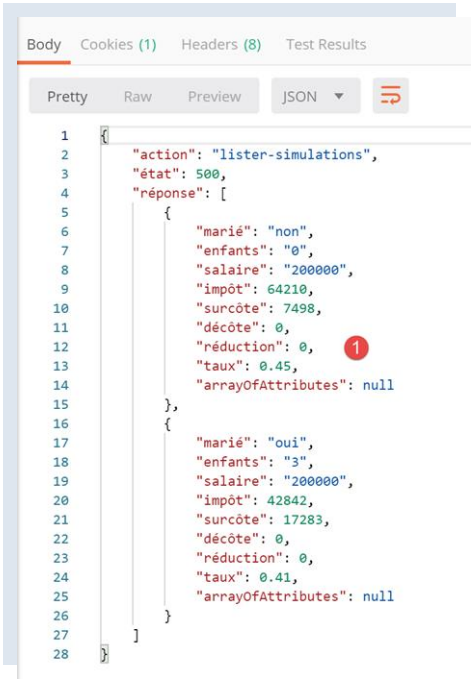
Maintenant une requête avec un n° de simulation qui n'existe pas :



Ci-dessus :

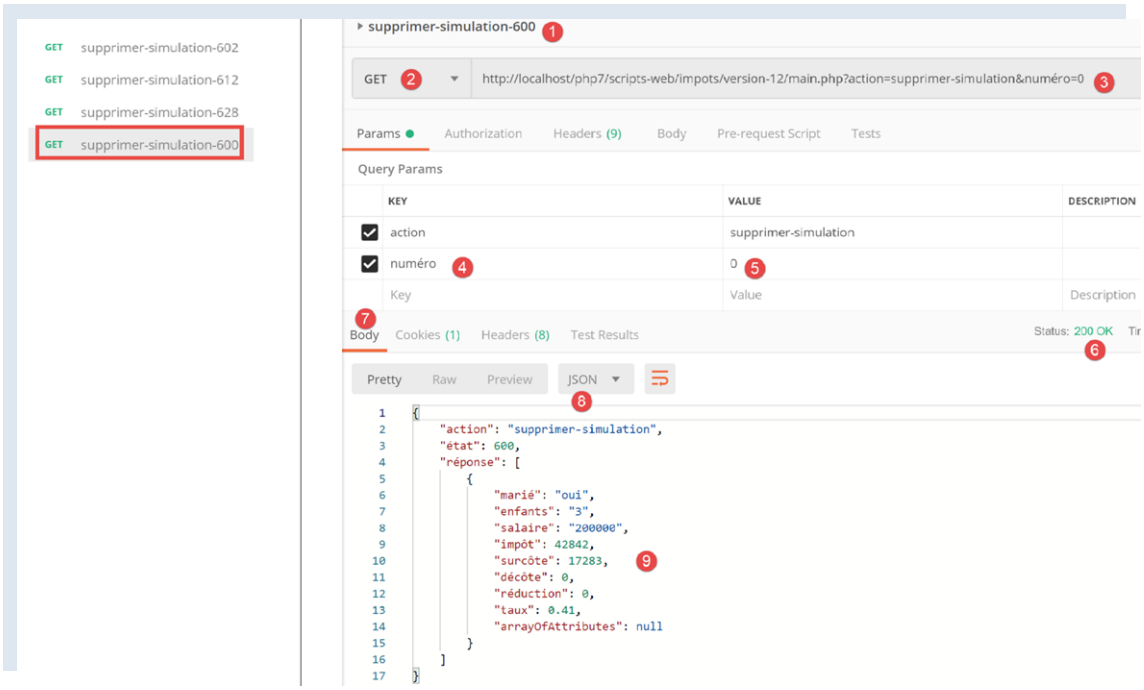
- en **[1-5]**, une requête avec un n° de simulation égal à 100 qui n'existe pas dans la liste des simulations ;
- en **[6-9]**, la réponse jSON du serveur ;

Maintenant, nous allons supprimer la simulation n° 0 de la liste, donc la première simulation. Tout d'abord redemandons cette liste avec la requête **[lister-simulations-500]** :



- en **[1]**, il y a actuellement 2 simulations ;

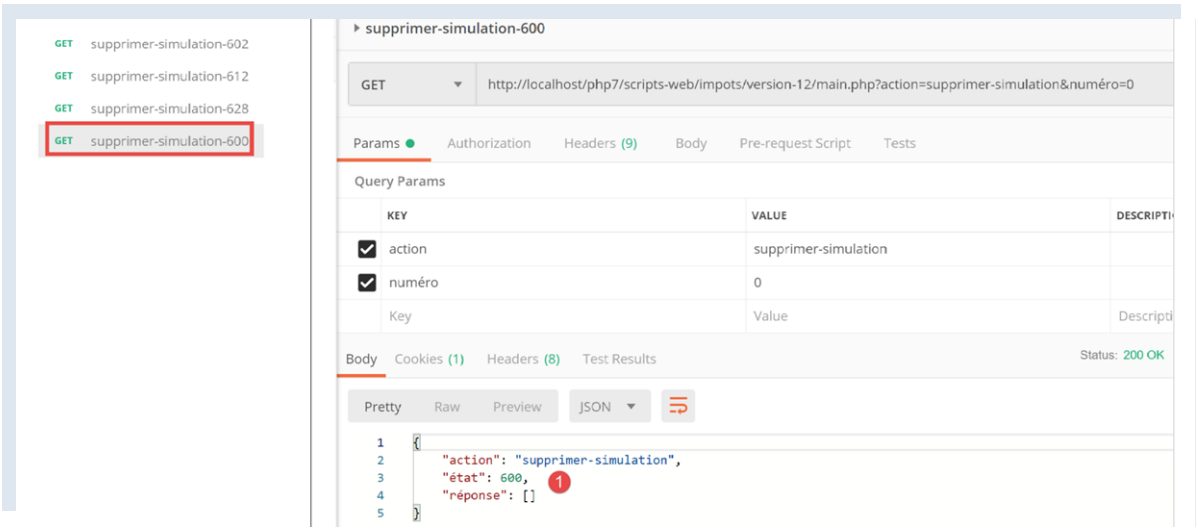
On supprime la 1re simulation (numéro 0) :



Ci-dessus :

- en **[1-5]**, on supprime la simulation n° 0 **[5]** ;
- en **[6-9]**, la réponse jSON du serveur. On voit que la simulation n° 0 a été supprimée ;

Répétons cette opération :



Ci-dessus :

- en **[1]**, il ne reste plus de simulations dans la session web du serveur ;

1.23.11.11 L'action `[fin-session]`

L'action **[fin-session]** est traitée par le contrôleur secondaire **[FinSessionController]** suivant :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
```

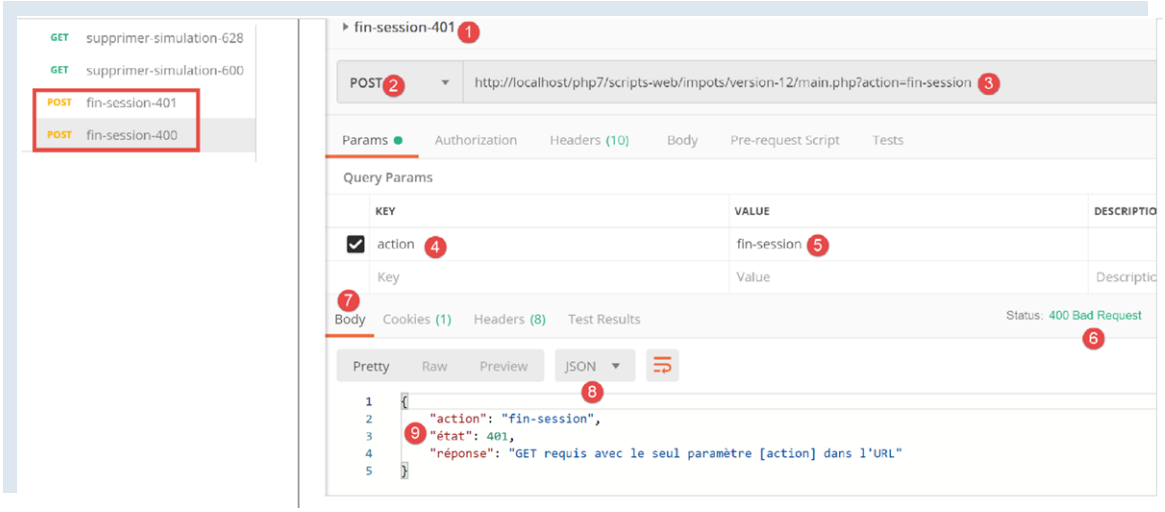
```
6.  use \Symfony\Component\HttpFoundation\Response;
7.  use \Symfony\Component\HttpFoundation\Request;
8.  use \Symfony\Component\HttpFoundation\Session\Session;
9.
10. class FinSessionController implements InterfaceController {
11.
12.   // $config est la configuration de l'application
13.   // traitement d'une requête Request
14.   // utile la session Session et peut la modifier
15.   // $infos sont des informations supplémentaires propres à chaque contrôleur
16.   // rend un tableau [$statusCode, $état, $content, $headers]
17.
18.   public function execute(
19.     array $config,
20.     Request $request,
21.     Session $session,
22.     array $infos = NULL): array {
23.
24.     // on doit avoir un unique paramètre GET
25.     $method = strtolower($request->getMethod());
26.     $erreur = $method !== "get" || $request->query->count() != 1;
27.     // erreur ?
28.     if ($erreur) {
29.       $état = 401;
30.       // résultat au contrôleur principal
31.       $message = "GET requis avec le seul paramètre [action] dans l'URL";
32.       return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
33.     }
34.
35.     // on mémorise le type de session
36.     $type = $session->get("type");
37.     // on invalide la session courante
38.     $session->invalidate();
39.     // on remet le type dans la nouvelle session
40.     $session->set("type", $type);
41.     // envoi de la réponse
42.     $état = 400;
43.     // résultat au contrôleur principal
44.     $content = ["réponse" => "session supprimée"];
45.     return [Response::HTTP_OK, $état, $content, []];
46.   }
47.
48. }
```

Commentaires

- requête **[GET main.php?action=fin-session]** ;
- lignes 25-33 : on vérifie que l'action est un GET avec l'unique paramètre **[fin-action]** ;
- ligne 38 : on invalide la session courante. Cela supprime les données enregistrées dans celle-ci et une nouvelle session est démarrée ;
- ligne 36 : avant la fin de la session, on mémorise le type **[json, xml, html]** de celle-ci ;
- ligne 40 : le type de la session précédente est replacé dans la nouvelle session. Finalement on repart avec une nouvelle session ayant l'uniqué clé **[type]** ;
- lignes 44-45 : on rend le résultat au contrôleur principal ;

1.23.11.12 Tests [Postman]

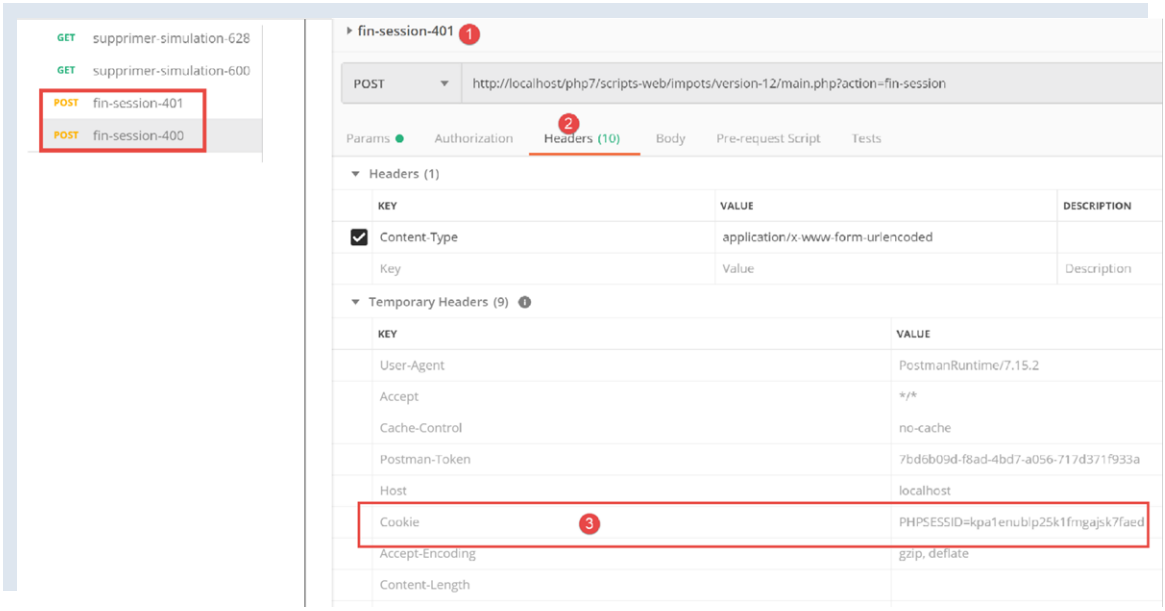
Nous allons faire un test d'erreur et un test de réussite :



Ci-dessus :

- en **[1-5]**, on demande la fin de session **[5]** avec un POST **[2]** au lieu du GET attendu ;
- en **[6-9]**, la réponse jSON du serveur ;

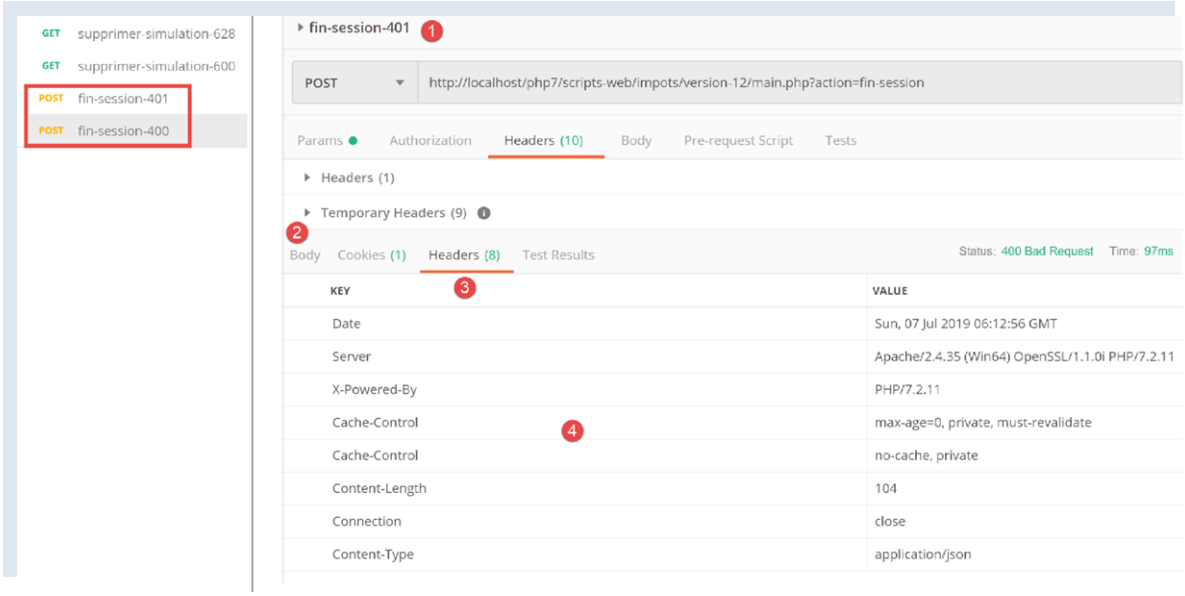
Maintenant un exemple de réussite. Regardons tout d'abord le cookie de session échangé entre le client **[Postman]** et le serveur lors du dernier test réalisé :



Ci-dessus :

- en **[3]**, le cookie de session envoyé par le client **[Postman]** au serveur ;

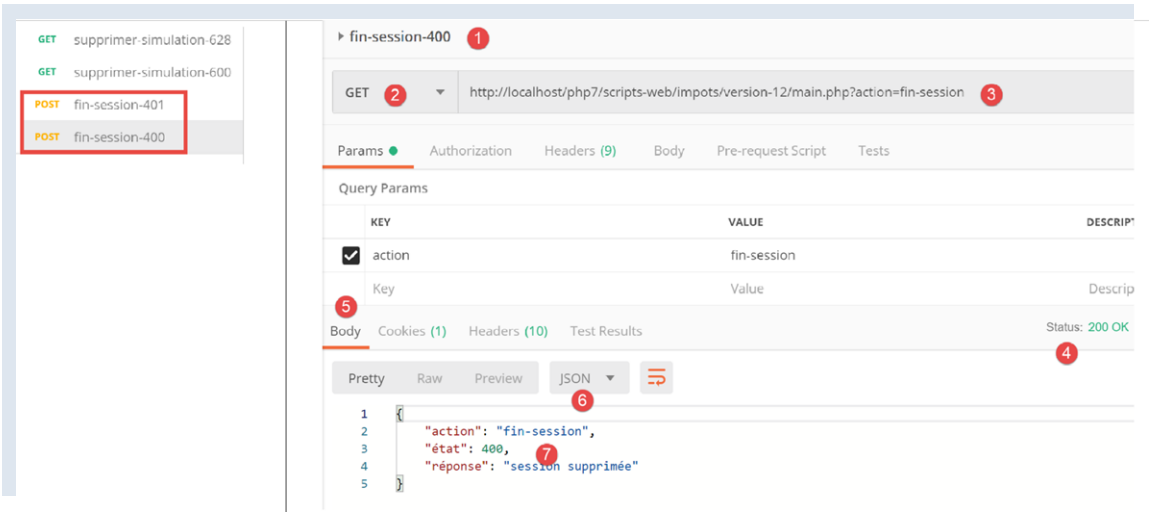
Regardons maintenant les entêtes HTTP envoyés par le serveur dans sa réponse :



Ci-dessus :

- en **[3-4]**, le cookie de session n'est pas dans la réponse du serveur. C'est normal. Celui-ci ne l'envoie qu'une fois : au début d'une nouvelles session web ;

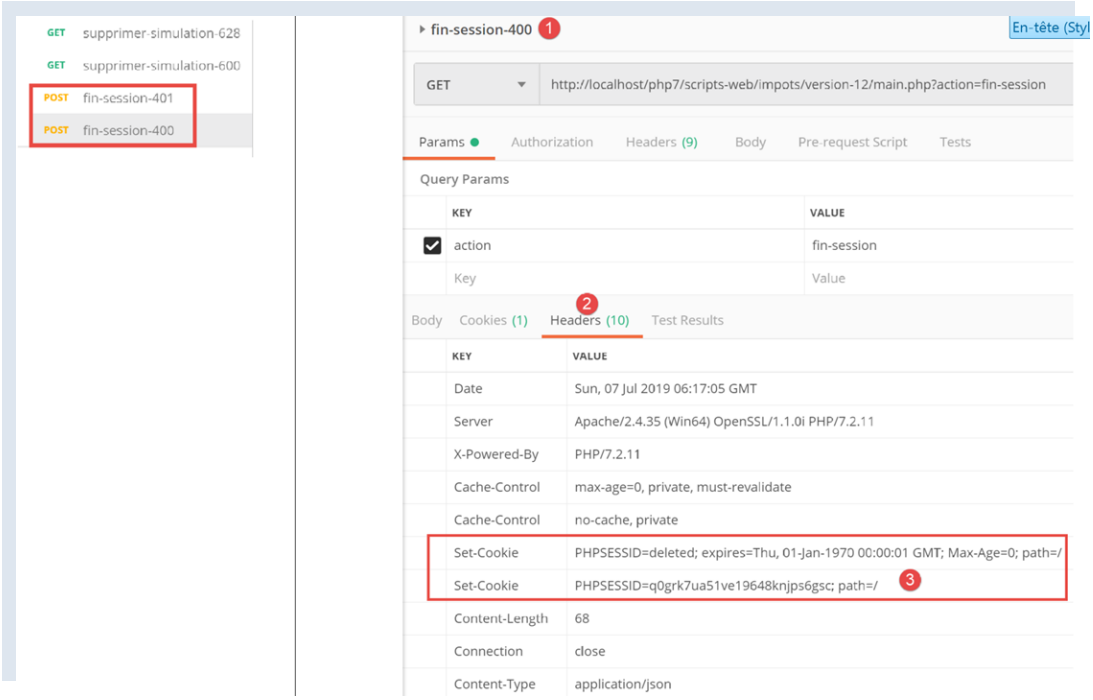
Maintenant exécutons une action **[fin-session]** valide :



Ci-dessus :

- en **[1-3]**, une action **[fin-session]** valide ;
- en **[4-7]**, la réponse jSON du serveur ;

Regardons les entêtes HTTP envoyés dans la réponse du serveur :

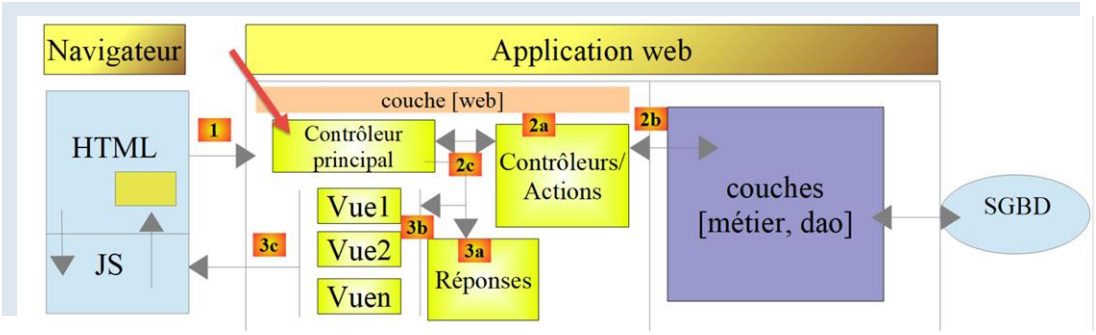


- en **[3]**, le serveur envoie l'entête **[Set-Cookie]** montrant par là qu'une nouvelle session web démarre ;

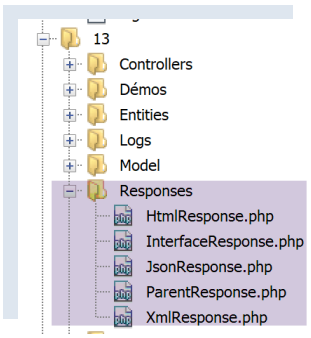
1.23.12 Les types de réponse du serveur

1.23.12.1 Introduction

Revenons sur l'architecture générale de l'application :



Nous allons présenter les types de réponse possibles **[3a]**. Celles-ci sont rassemblées dans le dossier **[Responses]** du projet :



Nous avons déjà présenté la classe **[JsonResponse]** au paragraphe lien. Elle implémente l'interface **[InterfaceResponse]** et étend la classe **[ParentResponse]**. C'est le cas des deux autres classes **[XmlResponse]** et **[HtmlResponse]**.

Rappelons la définition de l'interface **[InterfaceResponse]** :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Request;
7.  use Symfony\Component\HttpFoundation\Session\Session;
8.
9.  interface InterfaceResponse {
10.
11.   // Request $request : requête en cours de traitement
12.   // Session $session : la session de l'application web
13.   // array $config : la configuration de l'application
14.   // int statusCode : le code HTTP de statut de la réponse
15.   // array $content : la réponse du serveur
16.   // array $headers : les entêtes HTTP à ajouter à la réponse
17.   // Logger $logger : le logueur pour écrire des logs
18.
19.   public function send(
20.     Request $request = NULL,
21.     Session $session = NULL,
22.     array $config,
23.     int $statusCode,
24.     array $content,
25.     array $headers,
26.     Logger $logger = NULL): void;
27. }
```

- lignes 19-27 : l'interface **[InterfaceResponse]** a une unique méthode **[send]** pour envoyer la réponse au client ;
- lignes 11-17 : la signification des différents paramètres de la méthode **[send]** ;
- lignes 23-25 : les paramètres **[$statusCode, $content, $headers]** sont la réponse standard des contrôleurs secondaires de l'application. Cependant la réponse peut avoir besoin d'autres informations. Aussi lui donne-t-on les trois premiers paramètres (lignes 20-22) qui lui donnent accès à la totalité des informations concernant la requête, la session, la configuration ;
- ligne 26 : la réponse a besoin du **[Logger]** car elle va loguer la réponse envoyée au client ;

Rappelons maintenant le code de la classe **[ParentResponse]**, classe parent des trois types de réponse qui factorise ce qui leur est commun : l'envoi effectif d'une réponse texte au client :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Response;
7.
8.  class ParentResponse {
9.
10.   // int $statusCode : le code HTTP de statut de la réponse
11.   // string $content : le corps de la réponse à envoyer
12.   // selon les cas, c'est une chaîne jSON, XML, HTML
13.   // array $headers : les entêtes HTTP à ajouter à la réponse
14.
15.   public function sendResponse(
16.     int $statusCode,
17.     string $content,
18.     array $headers): void {
19.
20.     // préparation de la réponse texte du serveur
21.     $response = new Response();
22.     $response->setCharset("utf-8");
23.     // code de statut
24.     $response->setStatusCode($statusCode);
25.     // headers
26.     foreach ($headers as $text => $value) {
27.       $response->headers->set($text, $value);
28.     }
29.     // on envoie la réponse
30.     $response->setContent($content);
31.     $response->send();
32.   }
33. }
```

Commentaires

- lignes 10-13 : la signification des trois paramètres de la méthode **[send]** ;
- ligne 17 : on notera que le corps de la réponse est de type **[string]** et donc prêt à être envoyé (ligne 30) ;
- ligne 22 : la réponse aura des caractères UTF-8 ;
- ligne 24 : code de statut HTTP de la réponse ;
- lignes 26-28 : ajout des entêtes HTTP donnés par le code appelant ;
- lignes 30-31 : envoi de la réponse au client ;

Rappelons enfin le code du contrôleur principal qui demande l'envoi de la réponse au client :

```
1.  // on ajoute les clés [action, état] à la réponse du contrôleur
2.  $content = ["action" => $action, "état" => $état] + $content;
3.  // on instancie l'objet [Response] chargée d'envoyer la réponse au client
4.  $response = __NAMESPACE__ . $config["types"][$type]["response"];
5.  (new $response())->send($request, $session, $config, $statusCode, $content, $headers, $logger);
6.
7.  // la réponse a été envoyée - on libère les ressources
8.  $logger->close();
9.  exit;
```

- ligne 4 : on fixe le nom de la classe **[Response]** à instancier ;
- ligne 5 : on l'instancie et on envoie la réponse au client grâce à la méthode **[send($request, $session, $config, $statusCode, $content, $headers, $logger)]**. Parce qu'elles implémentent la même interface **[InterfaceResponse]**, les méthodes **[send]** des différents types de réponse ont toutes la même signature ;

1.23.12.2 La classe [JsonResponse]

Elle a déjà présentée au paragraphe lien. Nous redonnons cependant son code pour mieux souligner l'homogénéité des trois classes de réponse :

La classe **[JsonResponse]** implémente l'interface **[InterfaceResponse]** de la façon suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\Serializer\Encoder\JsonEncode;
7.  use Symfony\Component\Serializer\Encoder\JsonEncoder;
8.  use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;
9.  use Symfony\Component\Serializer\Serializer;
10. use \Symfony\Component\HttpFoundation\Request;
11. use \Symfony\Component\HttpFoundation\Session\Session;
12.
13. class JsonResponse extends ParentResponse implements InterfaceResponse {
14.
15.   // Request $request : requête en cours de traitement
16.   // Session $session : la session de l'application web
17.   // array $config : la configuration de l'application
18.   // int statusCode : le code HTTP de statut de la réponse
19.   // array $content : la réponse du serveur
20.   // array $headers : les entêtes HTTP à ajouter à la réponse
21.   // Logger $logger : le logueur pour écrire des logs
22.
23.   public function send(
24.     Request $request = NULL,
25.     Session $session = NULL,
26.     array $config,
27.     int $statusCode,
28.     array $content,
29.     array $headers,
30.     Logger $logger = NULL): void {
31.
32.     // préparation sérialiseur symfony
33.     $serializer = new Serializer(
34.       [
35.       // nécessaire pour la sérialisation d'objets
36.       new ObjectNormalizer()],
37.       // encodeur jSON
38.       // pour les options, faire des OU entre les différentes options
39.       [new JsonEncoder(new JsonEncode([JsonEncode::OPTIONS => JSON_UNESCAPED_UNICODE]))]
40.     );
```

```
41.     // sérialisation jSON
42.     $json = $serializer->serialize($content, 'json');
43.     // headers
44.     $headers = array_merge($headers, ["content-type" => "application/json"]);
45.     // envoi réponse
46.     parent::sendResponse($statusCode, $json, $headers);
47.     // log
48.     if ($logger !== NULL) {
49.       $logger->write("réponse=$json\n");
50.     }
51.   }
52.
53. }
```

Commentaires

- ligne 13 : la classe implémente l'interface **[InterfaceResponse]** ;
- ligne 13 : la classe étend la classe **[ParentResponse]**. Tous les types de **[Response]** étendent cette classe. C'est cette classe parent qui envoie la réponse au client (ligne 46). C'est parce que ce code était commun à tous les types de **[Response]** qu'il a été factorisé dans une classe parent ;
- lignes 33-40 : instanciation du sérialiseur **[Symfony]** qui va traduire la réponse du serveur **[$content]** en chaîne jSON (ligne 42) ;
- lignes 34-36 : le 1er paramètre du constructeur de **[Serializer]** est un tableau. Dans celui-ci, on met une instance de la classe **[ObjectNormalizer]** nécessaire à la sérialisation d'objets. Ce cas se présente dans cette application avec une liste de simulations où chaque simulation est une instance de la classe **[Simulation]** ;
- ligne 39 : le second paramètre du constructeur de **[Serializer]** est également un tableau : on y met tous les encodeurs utilisés dans une sérialisation (XML, jSON, CSV…) ;
- ligne 39 : il n'y aura qu'un encodeur ici, de type **[JsonEncoder]**. Le constructeur sans paramètres aurait pu être suffisant. Ici, nous avons passé un paramètre **[JsonEncode]** au constructeur, uniquement pour passer des options d'encodage jSON ;
- ligne 39 : le paramètre du constructeur **[JsonEncode]** est un tableau d'options. Ici on utilise l'option **[JSON_UNESCAPED_UNICODE]** pour demander que les caractères UTF-8 de la chaîne jSON soient rendus nativement et non pas « échappés » ;
- ligne 42 : le corps de la réponse HHTP est sérialisé en jSON grâce au sérialiseur précédent ;
- ligne 44 : on ajoute l'entête HTTP qui dit au client qu'on va lui envoyer du jSON ;
- ligne 46 : on demande à la classe parent d'envoyer la réponse au client ;
- lignes 48-50 : on logue la réponse jSON ;

1.23.12.3 La classe [XmlResponse]

La classe **[XmlResponse]** implémente l'interface **[InterfaceResponse]** de la façon suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Request;
7.  use Symfony\Component\HttpFoundation\Session\Session;
8.  use Symfony\Component\Serializer\Encoder\JsonEncode;
9.  use Symfony\Component\Serializer\Encoder\JsonEncoder;
10. use Symfony\Component\Serializer\Encoder\XmlEncoder;
11. use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;
12. use Symfony\Component\Serializer\Serializer;
13.
14. class XmlResponse extends ParentResponse implements InterfaceResponse {
15.
16.   // Request $request : requête en cours de traitement
17.   // Session $session : la session de l'application web
18.   // array $config : la configuration de l'application
19.   // int statusCode : le code HTTP de statut de la réponse
20.   // array $content : la réponse du serveur
21.   // array $headers : les entêtes HTTP à ajouter à la réponse
22.   // Logger $logger : le logueur pour écrire des logs
23.
24.   public function send(
25.     Request $request = NULL,
26.     Session $session = NULL,
27.     array $config,
28.     int $statusCode,
29.     array $content,
30.     array $headers,
31.     Logger $logger = NULL): void {
32.
```

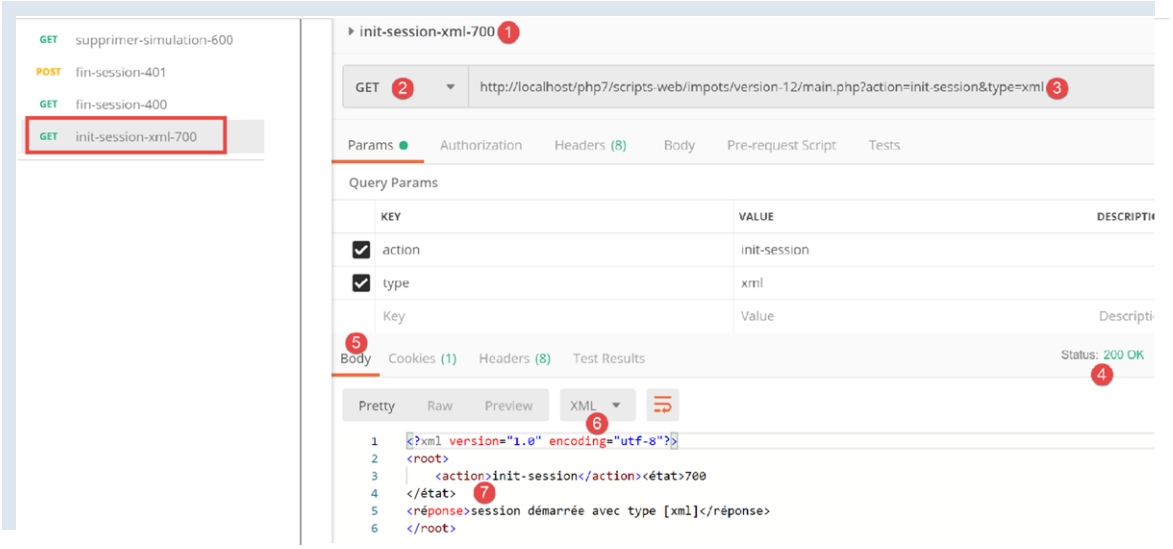
```
33.     // préparation sérialiseur symfony
34.     $serializer = new Serializer(
35.       // nécessaire pour la sérialisation d'objets
36.       [new ObjectNormalizer()],
37.       [
38.       // sérialisation XML
39.       new XmlEncoder(
40.         [
41.         XmlEncoder::ROOT_NODE_NAME => 'root',
42.         XmlEncoder::ENCODING => 'utf-8'
43.         ]
44.       ),
45.       // sérialisation jSON
46.       new JsonEncoder(new JsonEncode([JsonEncode::OPTIONS => JSON_UNESCAPED_UNICODE]))
47.       ]
48.     );
49.     // sérialisation XML
50.     $xml = $serializer->serialize($content, 'xml');
51.     // headers
52.     $headers = array_merge($headers, ["content-type" => "application/xml"]);
53.     // envoi réponse
54.     parent::sendResponse($statusCode, $xml, $headers);
55.     // log
56.     if ($logger !== NULL) {
57.       // log en jSON
58.       $log = $serializer->serialize($content, 'json');
59.       $logger->write("réponse=$log\n");
60.     }
61.   }
62.
63. }
```

Commentaires

- lignes 34-48 : instanciation d'un sérialiseur Symfony. Le constructeur admet deux paramètres de type tableau ;
- ligne 36 : le 1er tableau comprend une instance de type **[ObjectNormalizer]** qui intervient dans la sérialisation d'objets ;
- lignes 37-47 : le second tableau contient les encodeurs utilisés pour la sérialisation. On peut prévoir divers types de sérialisation avec le même sérialiseur ;
- lignes 38-44 : l'encodeur XML ;
- ligne 41 : on fixe la racine du code XML généré. Celui-ci aura la forme <root>**[autres balises XML]**</root> ;
- ligne 42 : l'encodage utilisera des caractères UTF-8 ;
- ligne 46 ; l'encodeur jSON. Celui-ci sera utilisé pour le log de la réponse dans le fichier **[logs.txt]** qui est fait en jSON ;
- ligne 50 : le corps de la réponse envoyée au client est sérialisée en XML ;
- ligne 52 : on ajoute aux entêtes reçus en paramètre (ligne 30) l'entête HTTP qui indique au client qu'on lui envoie un document XML ;
- ligne 54 : envoi effectif de la réponse au client par la classe parent ;
- lignes 56-60 : log en jSON de la réponse ;

1.23.12.4 Tests [Postman]

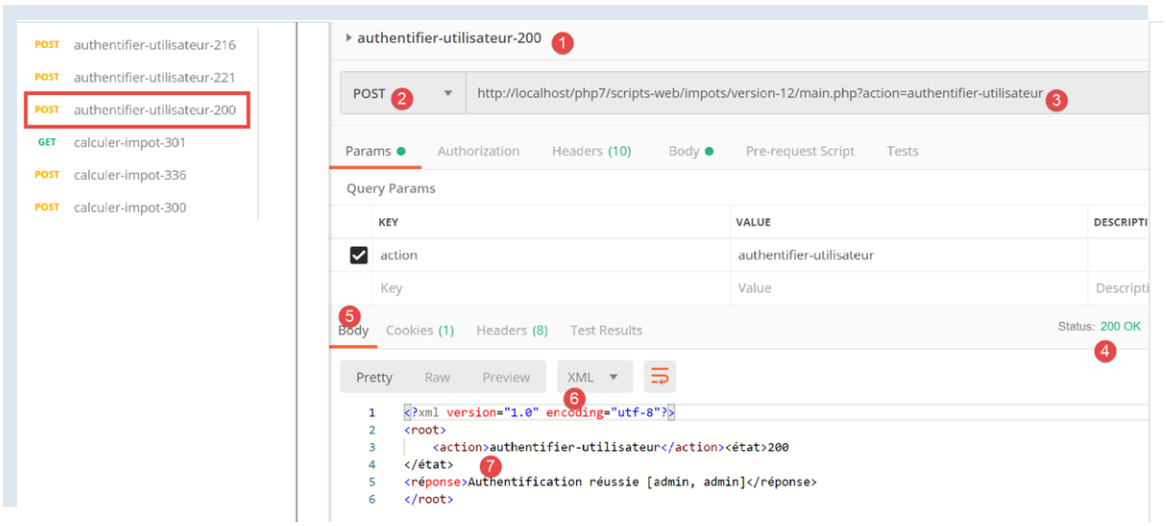
Nous avons déjà fait tous les tests d'erreurs possibles en jSON. Il n'y a rien à faire de plus en XML. Nous montrons deux exemples de réponse XML :



Ci-dessus :

- en **[1-3]**, la requête de début de session XML ;
- en **[4-7]**, la réponse XML du serveur ;

A partir de maintenant, toutes les réponses du serveur seront en XML. On peut reprendre toutes les requêtes déjà utilisées dans **[Postman]** sans les changer et on aura pour chacune d'elles une réponse XML. Faisons par exemple une authentification réussie :



Ci-dessus :

- en **[1-3]**, une demande d'authentification valide ;
- en **[4-7]**, la réponse XML du serveur ;

1.23.12.5 La réponse [HtmlResponse]

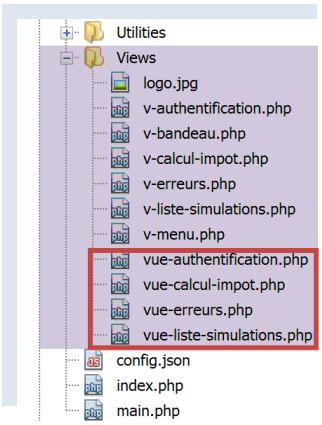
Lorsque le type de la session est **[html]** un objet de type **[HtmlResponse]** est instancié pour envoyer la réponse au client. Celle-ci va envoyer au client un flux HTML qui dépend du code d'état rendu par le contrôleur secondaire qui a traité l'action. Cette correspondance **[état=>vue]** est inscrite dans le fichier de configuration **[config.json]** de la façon suivante :

```
1.  "vues": {
2.  "vue-authentification.php": [700, 221, 400],
3.  "vue-calcul-impot.php": [200, 300, 341, 350, 800],
4.  "vue-liste-simulations.php": [500, 600]
5.  },
6.  "vue-erreurs": "vue-erreurs.php"
```

Cette configuration se lit de la façon suivante : **['nom de la vue' => 'états associés à cette vue']**

- ligne 2 : si le contrôleur secondaire a rendu un état du tableau **[700, 221, 400]**, alors il faut afficher la vue **[vue-authentification.php]** ;
- ligne 3 : si le contrôleur secondaire a rendu un état du tableau **[200, 300, 341, 350, 800]**, alors il faut afficher la vue **[vue-calcul-impot.php]** ;
- ligne 4 : si le contrôleur secondaire a rendu un état du tableau **[500, 600]**, alors il faut afficher la vue **[vue-liste-simulations.php]** ;
- ligne 6 : si le contrôleur secondaire a rendu un état qui n'est dans aucun des tableaux précédents, alors il faut afficher la vue **[vue-erreurs.php]** ;

Les vues sont rassemblées dans le dossier **[Views]** du projet :



Le code de la classe **[HtmlResponse]** est le suivant :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Request;
7.  use Symfony\Component\HttpFoundation\Session\Session;
8.  use Symfony\Component\Serializer\Encoder\JsonEncode;
9.  use Symfony\Component\Serializer\Encoder\JsonEncoder;
10. use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;
11. use Symfony\Component\Serializer\Serializer;
12.
13. class HtmlResponse extends ParentResponse implements InterfaceResponse {
14.
15.   // Request $request : requête en cours de traitement
16.   // Session $session : la session de l'application web
17.   // array $config : la configuration de l'application
18.   // int statusCode : le code HTTP de statut de la réponse
19.   // array $content : la réponse du serveur
20.   // array $headers : les entêtes HTTP à ajouter à la réponse
21.   // Logger $logger : le logueur pour écrire des logs
22.
23.   public function send(
24.     Request $request = NULL,
25.     Session $session = NULL,
26.     array $config,
27.     int $statusCode,
28.     array $content,
29.     array $headers,
30.     Logger $logger = NULL): void {
31.
32.     // préparation sérialiseur symfony
33.     $serializer = new Serializer(
34.       [
35.       // pour la sérialisation des objets
36.       new ObjectNormalizer()],
37.       [
```

```
38.       // pour la sérialisation jSON du log de la réponse
39.       new JsonEncoder(new JsonEncode([JsonEncode::OPTIONS => JSON_UNESCAPED_UNICODE]))
40.       ]
41.     );
42.     // la réponse HTML dépend du code d'état rendu par le contrôleur
43.     $état = $content["état"];
44.     // à un état, correspond une vue - on cherche celle-ci dans la configuration de l'application
45.     // la liste des vues
46.     $vues = array_keys($config["vues"]);
47.     $trouvé = false;
48.     $i = 0;
49.     // on parcourt la liste des vues
50.     while (!$trouvé && $i < count($vues)) {
51.       // états associés à la vue n° i
52.       $états = $config["vues"][$vues[$i]];
53.       // est-ce que l'état cherché se trouve dans les états associés à la vue n° I ?
54.       if (in_array($état, $états)) {
55.         // la vue affichée sera la vue n° i
56.         $vueRéponse = $vues[$i];
57.         $trouvé = true;
58.       }
59.       // vue suivante
60.       $i++;
61.     }
62.     // trouvé ?
63.     if (!$trouvé) {
64.       // si aucune vue n'existe pour l'état actuel de l'application
65.       // on rend la vue d'erreurs
66.       $vueRéponse = $config["vue-erreurs"];
67.     }
68.     // on récupère la vue HTML à afficher dans une chaîne de caractères
69.     ob_start();
70.     require __DIR__ . "/../Views/$vueRéponse";
71.     $html = ob_get_clean();
72.     // on indique dans les entêtes qu'on va envoyer du HTML
73.     $headers = array_merge($headers, ["content-type" => "text/html"]);
74.     // la classe parent s'occupe de l'envoi effectif de la réponse
75.     parent::sendResponse($statusCode, $html, $headers);
76.     // log en jSON de la réponse sans le HTML
77.     if ($logger !== NULL) {
78.       // log en jSON de la réponse du contrôleur secondaire qui a traité l'action
79.       $log = $serializer->serialize($content, 'json');
80.       $logger->write("réponse=$log\n");
81.     }
82.   }
83.
84. }
```

Commentaires

- lignes 32-41 : on instancie un sérialiseur Symfony. Celui-ci est nécessaire pour le log jSON de la réponse du contrôleur qui a traité l'action (lignes 72-82) ;
- lignes 42-57 : on cherche dans la configuration de l'application, la vue qui doit être affichée. Celle-ci dépend du code d'état rendu par le contrôleur qui a traité l'action. Ce code est dans **[$content['état']]** (ligne 43) ;
- lignes 42-61 : on cherche la vue qui correspond à cet état ;
- lignes 62-67 : si aucune vue n'a été trouvée alors on est dans une situation d'un code d'état anormal pour l'application HTML. On explicitera plus loin cette notion d'états anormaux. Dans ce cas, on affiche une vue d'erreur ;
- lignes 68-70 : on interprète le code PHP de la vue sélectionnée et on met le résultat dans la variable **[$html]** (ligne 71) ;
- ce code mérite quelques explications. Imaginons que la vue sélectionnée soit **[vue-authentification.php]** qui présente un formulaire web d'authentification :
  - ligne 69 : la fonction **[ob_start]** démarre ce que la documentation appelle une temporisation de sortie. Tout ce qui est écrit par des opérations print, require… et qui normalement est immédiatement envoyé au client, va dans un tampon de sortie (ob=output buffer) sans être envoyé au client ;
  - ligne 70 : on charge la vue **[vue-authentification.php]** qui est une vue HTML dynamique contenant du code PHP. Se passent alors deux choses :
    - le code PHP de la vue **[vue-authentification.php]** est chargé et interprété. Le résultat est une vue qu'on appellera **[vue-authentification.html]** qui ne contient que du code HTML, voire CSS et Javascript mais plus de PHP ;
    - ce code HTML est normalement envoyé au client. C'est en fait le cas de tout texte rencontré par l'interpréteur PHP et qui n'est pas du code PHP. A cause de la temporisation de sortie, ce code HTML est mis dans le tampon de sortie sans être envoyé au client ;
  - ligne 71 : la fonction **[ob_get_clean]** fait deux choses :

- elle met dans la variable **[$html]** le contenu du buffer de sortie, donc la page **[vue-authentification.html]** qu'on y a mise ;
- elle vide le buffer de sortie. Pour celui-ci, tout se passe comme si rien ne s'était produit. Par ailleurs, le client n'a toujours rien reçu ;

- ligne 70 : on est ici en cours d'exécution de la classe **[HtmlResponse]** qui se trouve dans le dossier **[Responses]**. Pour touver la vue, il faut donc remonter d'un niveau **[..]** puis passer dans le dossier **[Views]**. **[__DIR__]** est le nom absolu du dossier dans lequel se trouve le script en cours d'exécution, dans notre exemple le dossier **[C:/myprograms/laragon-lite/www/php7/scripts-web/impots/13/Responses]** ;
- ligne 73 : on ajoute aux entêtes HTTP reçus en paramètre (ligne 29), l'entête qui dit au client qu'on va lui envoyer du HTML ;
- ligne 75 : on demande à la classe parent de procéder à l'envoi effectif de la réponse au client ;
- lignes 77-81 : on logue en jSON la réponse **[$content]** fournie par le contrôleur secondaire qui a traité l'action en cours ;

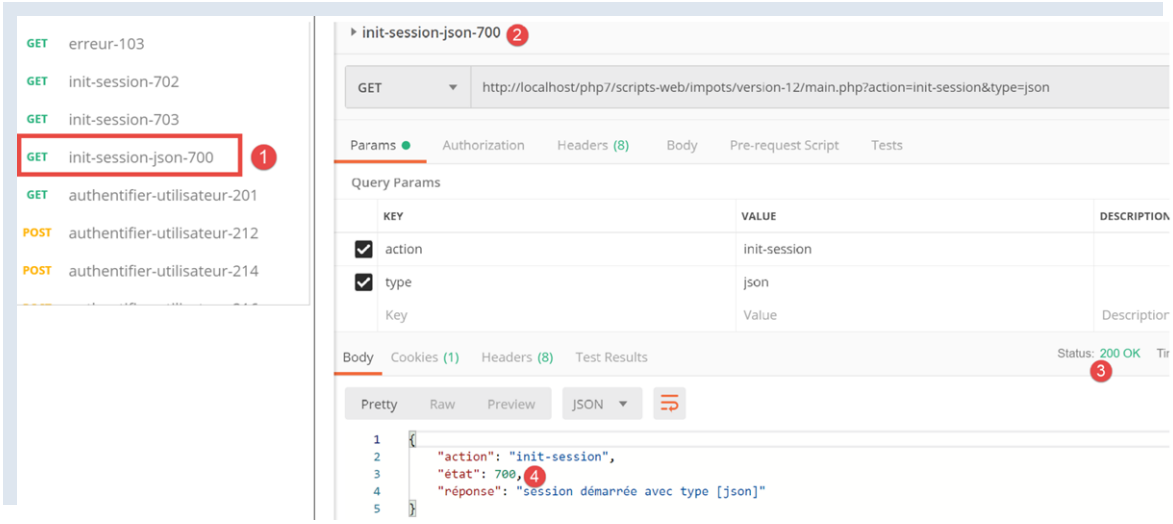
1.23.12.6 Tests [Postman]

Pour tester véritablement le mode HTML de la session, il nous faudrait passer en revue toutes les vues. Nous allons le faire ultérieurement. Nous allons faire le test suivant :

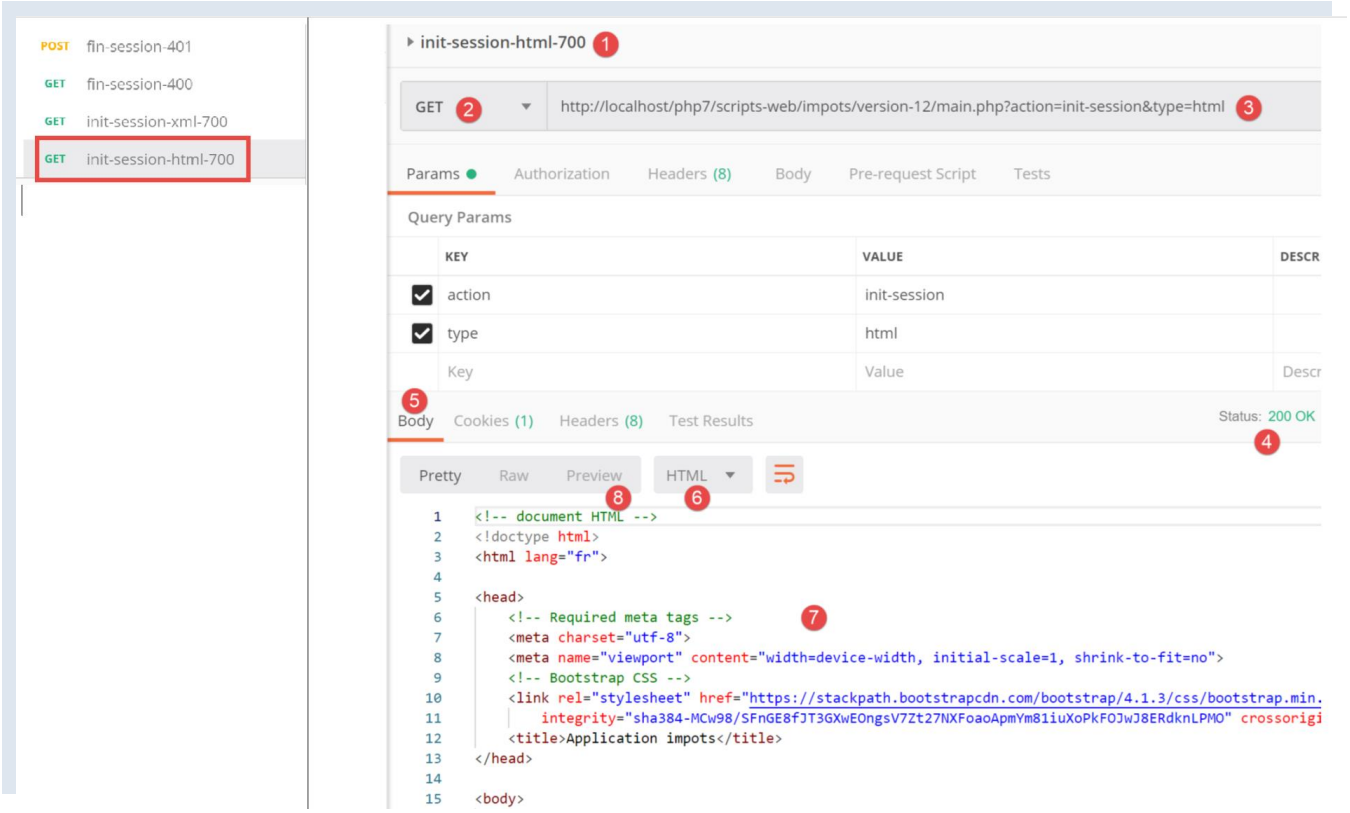
Regardons la liste des vues dans le fichier de configuration :

```
1.  "vues": {
2.          "vue-authentification.php": [700, 221, 400],
3.          "vue-calcul-impot.php": [200, 300, 341, 350, 800],
4.          "vue-liste-simulations.php": [500, 600]
5.      },
6.  "vue-erreurs": "vue-erreurs.php"
```

On peut trouver le contexte produisant certains des codes d'état ci-dessus en regardant les tests **[Postman]** réalisés :

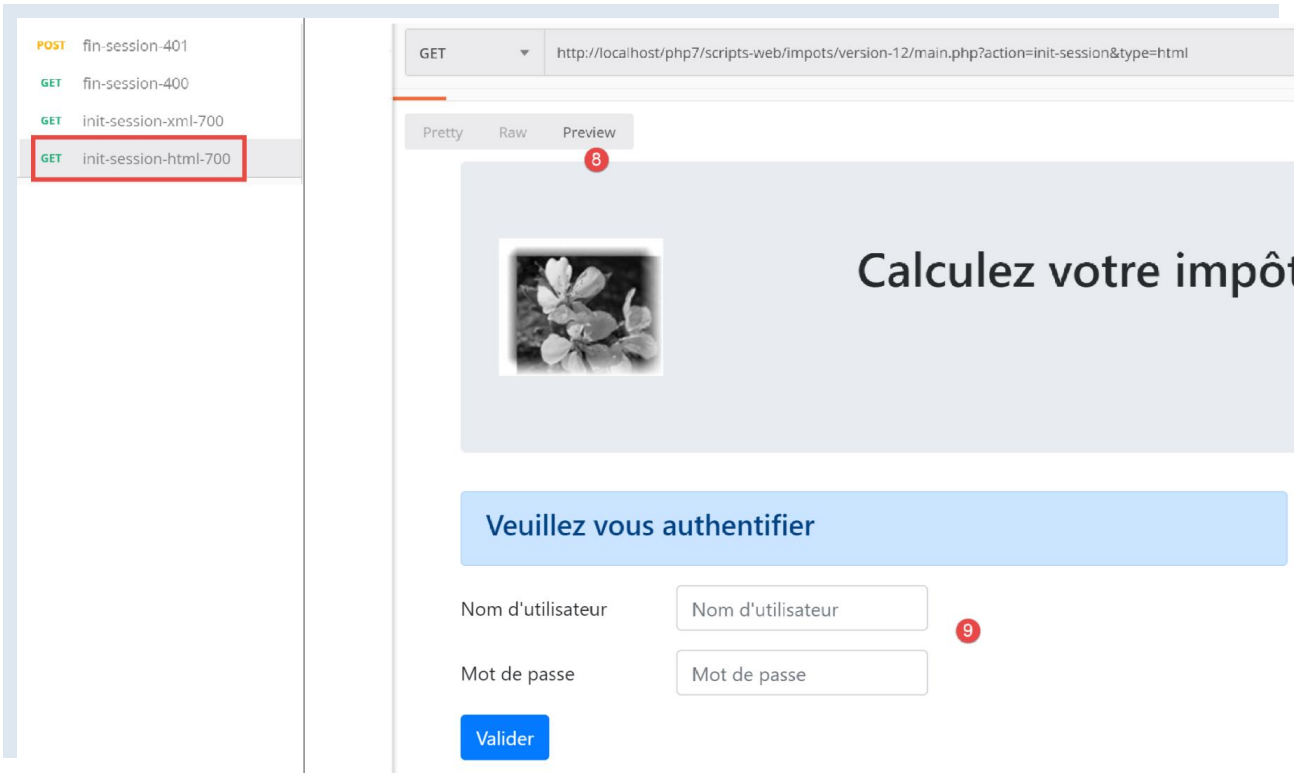


On voit que le code d'état **[700]** est celui d'une action **[init-session]** réussie **[2]**. Ci-dessus, on a une réponse jSON mais elle peut être de type XML ou HTML. C'est ce dernier cas qui va être testé. D'après le fichier de configuration, c'est la vue **[vue-authentification.php]** qui constitue la réponse HTML. Vérifions.



Ci-dessus :

- en **[1-3]**, on initialise une session HTML. On attend donc une réponse HTML ;
- en **[4-8]**, la réponse HTML du serveur ;
- l'onglet **[8]** permet d'avoir une prévisualisation du code HTML reçu ;



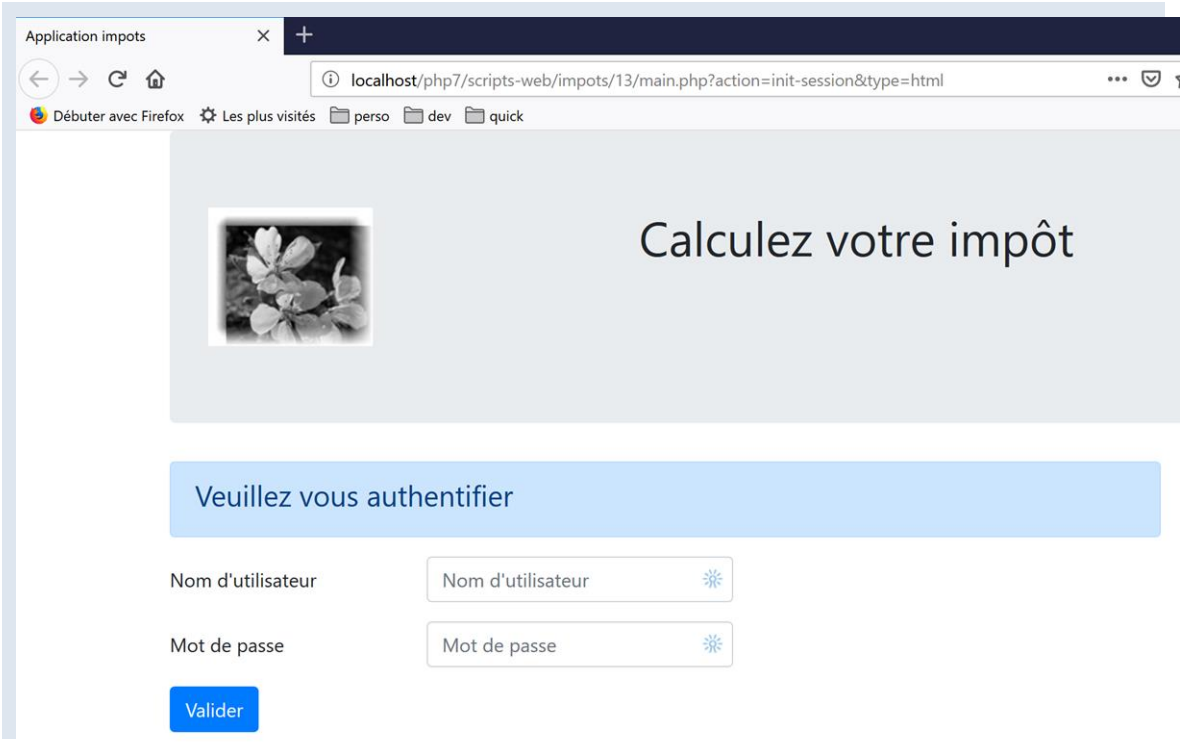
- en **[8-9]**, un aperçu de la vue HTML ;

1.23.13 L'application web HTML

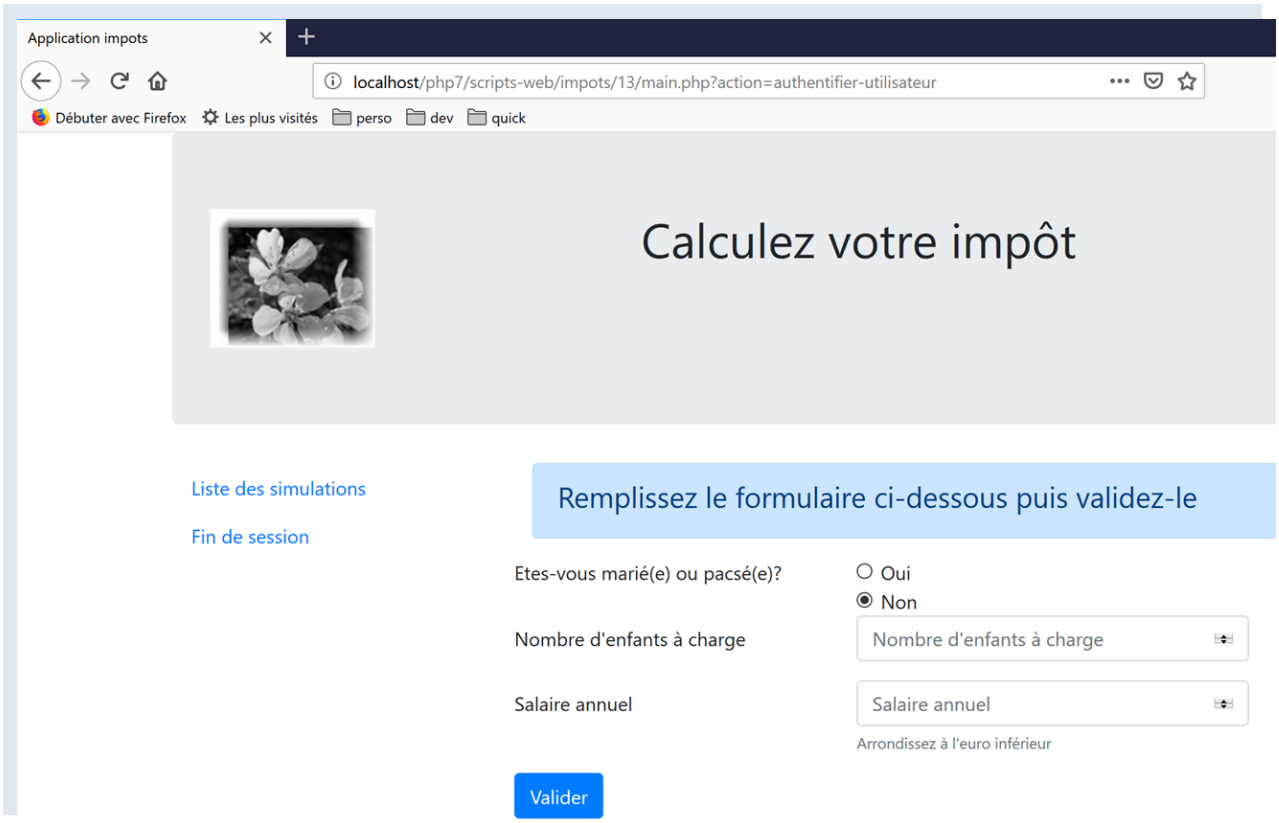
1.23.13.1 Présentation des vues

L'application web HTML utilisera quatre vues :

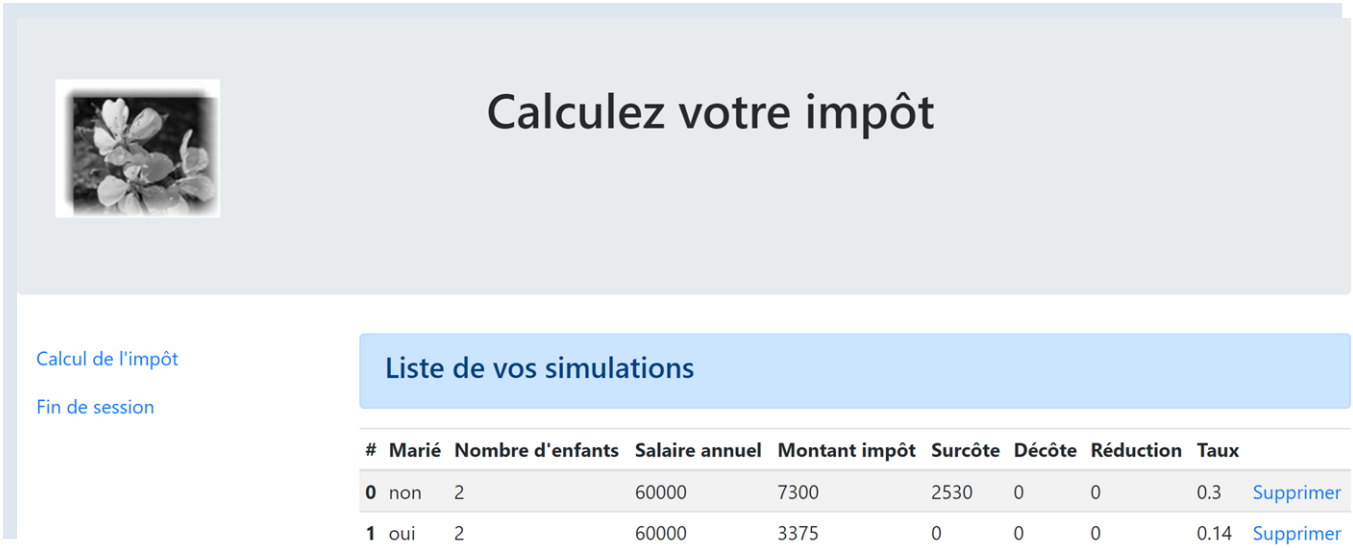
La vue d'authentification :



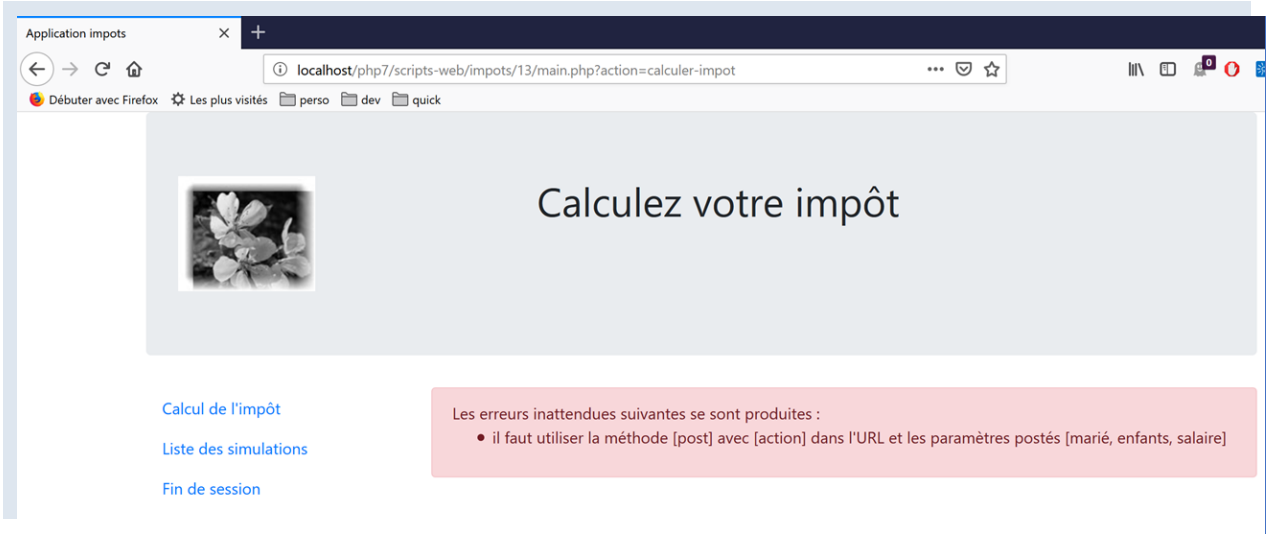
La vue de calcul de l'impôt :



La vue de la liste des simulations :



La vue des erreurs inattendues :

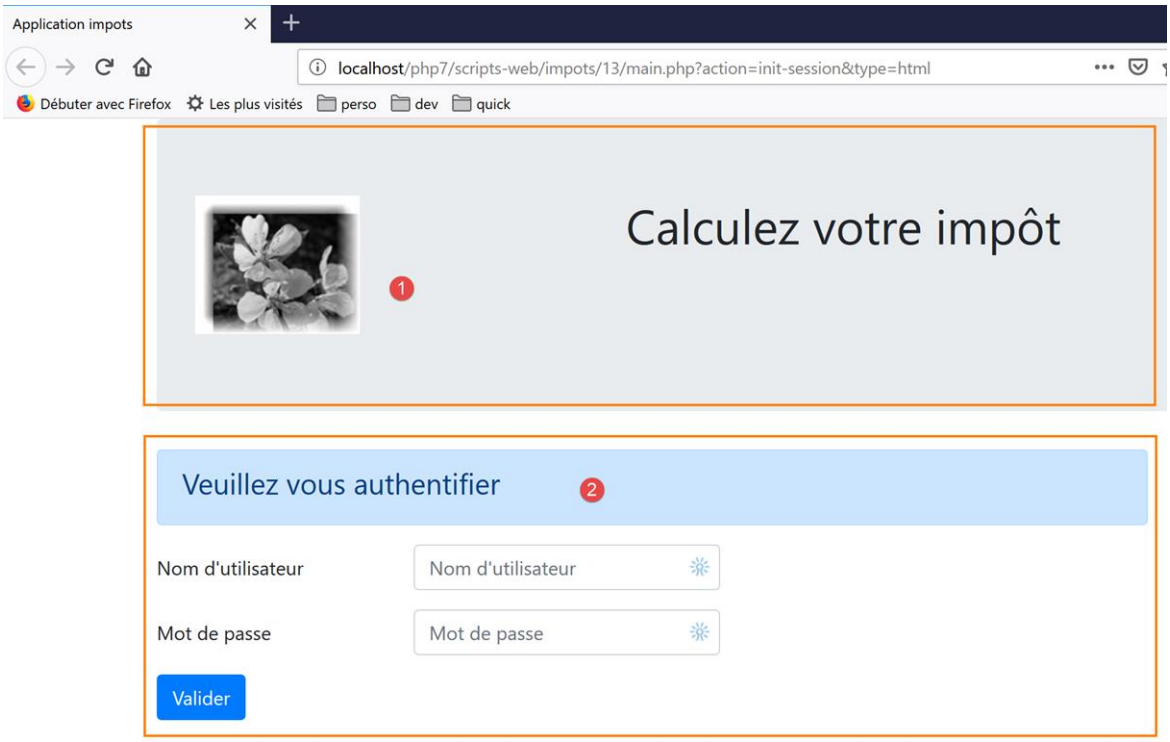


Nous allons décrire ces vues l'une après l'autre.

1.23.13.2 La vue d'authentification

1.23.13.2.1 Présentation de la vue

La vue d'authentification est la suivante :



La vue est composée de deux éléments qu'on appellera des fragments :

- le fragment **[1]** est généré par un script **[v-bandeau.php]** ;
- le fragment **[2]** est généré par un script **[v-authentification.php]** ;

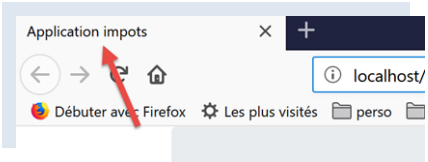
La vue d'authentification est générée par la page **[vue-authentification.php]** suivante :

```
1.  <?php
2.  // données de test de la page
3.  // on encapsule les données de la pagé dans $page
4.  …
```

```
5.  ?>
6.
7.  <!doctype html>
8.  <html lang="fr">
9.      <head>
10.         <!-- Required meta tags -->
11.         <meta charset="utf-8">
12.         <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
13.         <!-- Bootstrap CSS -->
14.         <link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-
    MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
15.         <title>Application impots</title>
16.     </head>
17.     <body>
18.         <div class="container">
19.             <!-- bandeau sur 1 ligne et 12 colonnes -->
20.             <?php require "v-bandeau.php"; ?>
21.             <!-- formulaire d'authentification sur 9 colonnes -->
22.             <div class="row">
23.                 <div class="col-md-9">
24.                     <?php require "v-authentification.php" ?>
25.                 </div>
26.             </div>
27.             <?php
28.             // si erreur - on affiche une alerte d'erreur
29.             if ($modèle->error) {
30.               print <<<EOT
31.             <div class="row">
32.                 <div class="col-md-9">
33.                     <div class="alert alert-danger" role="alert">
34.                       Les erreurs suivantes se sont produites :
35.                       <ul>$modèle->erreurs</ul>
36.                     </div>
37.                 </div>
38.             </div>
39. EOT;
40.             }
41.             ?>
42.         </div>
43.     </body>
44. </html>
```

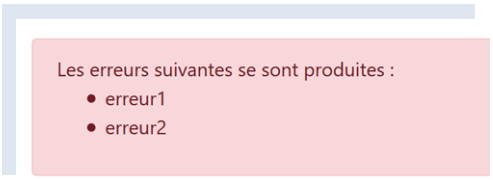
Commentaires

- ligne 7 : un document HTML commence avec cette ligne ;
- lignes 8-44 : la page HTML est encapsulée dans les balises <**html**> </**html**> ;
- lignes 9-16 : entête (**head**) du document HTML ;
- ligne 11 : la balise <**meta charset**> indique que le document est codé en UTF-8 ;
- ligne 12 : la balise <**meta name='viewport'**> fixe l'affichage initial de la vue : sur toute la largeur de l'écran qui l'affiche (width) à sa taille initiale (**initial-scale**) sans redimensionnement pour s'adapter à une taille plus petite d'écran (**shrink-to-fit**) ;
- ligne 14 : la balise <**link rel='stylesheet'**> fixe le fichier CSS qui gouverne l'apparence de la vue. Nous utilisons ici le framework CSS Bootstrap 4.1.3 **[https://getbootstrap.com/docs/4.0/getting-started/introduction/]** ;
- ligne 15 : la balise <**title**> fixe le titre de la page :



- lignes 17-43 : le corps de la page web est encapsulé dans les balises **<body></body>** ;
- lignes 18-42 : la balise délimite une section de la page affichée. Les attributs **[class]** utilisés dans la vue se réfèrent tous au framework CSS Bootstrap. La balise <**div class='container'**> délimite un conteneur Bootstrap ;
- ligne 20 : on inclut le script **[v-bandeau.php]**. Ce script génère le bandeau **[1]** de la page. Nous le décrirons bientôt ;
- lignes 22-26 : la balise <**div class='row'**> délimite une ligne Bootstrap. Ces lignes sont constituées de 12 colonnes ;
- ligne 23 : la balise <**div class='col-md-9'**> délimite une section de 9 colonnes ;
- ligne 24 : on inclut le script **[v-authentification.php]** qui affiche le formulaire d'authentification **[2]** de la page. Nous le décrirons bientôt ;

- ligne 27 : la balise **<?php** introduit du code PHP à l'intérieur de la page HTML. Ce code est exécuté avant l'affichage de la page HTML et peut modifier celle-ci ;
- ligne 29 : la totalité des données dynamique de la vue affichée sera encapsulée dans un objet **[$modèle]** de type **[stdClass]**. C'est un choix arbitraire. On aurait pu choisir un tableau associatif à la place pour le même résultat ;
- ligne 29 : l'authentification échoue si l'utilisateur entre des identifiants incorrects. Dans ce cas, la vue d'authentification est réaffichée avec un message d'erreur. L'attribut **[$modèle→error]** indique s'il faut afficher ce message d'erreur ;
- lignes 30-39 : cette syntaxe écrit tout le texte placé entre les symbole PHP <<<EOT (ligne 30 – on peut mettre ce qu'on veut à la place de EOT=End Of Text) et le symbole EOT de la ligne 39 (doit être identique au symbole utilisé ligne 30). Le symbole doit être écrit dans la 1re colonne de la ligne 39. Les variables PHP situées dans le texte entre les deux symboles EOT sont interprétées ;
- lignes 33-36 : délimitent une zone à fond rose (**class**="alert alert-danger") (ligne 33) ;



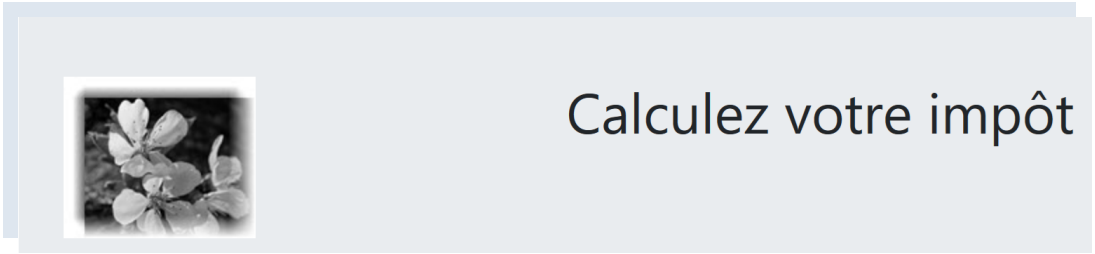
- ligne 34 : un texte ;
- ligne 35 : la balise HTML <**ul**> (unordered list) affiche une liste à puces. Chaque élément de la liste doit avoir la syntaxe **
élément
** ;

Retenons de ce code les éléments dynamiques à définir :

- **[$modèle→error]** : pour afficher un message d'erreur ;
- **[$modèle→erreurs]** : une liste (au sens HTML du terme) de messages d'erreur ;

1.23.13.2.2 Le fragment [v-bandeau.php]

Le fragment **[v-bandeau.php]** affiche le bandeau supérieur de toutes les vues de l'application web :



Le code du fragment **[v-bandeau.php]** est le suivant :

```
1.  <!-- Bootstrap Jumbotron -->
2.  <div class="jumbotron">
3.      <div class="row">
4.          <div class="col-md-4">
5.              <img src="<?= $logo ?>" alt="Cerisier en fleurs" />
6.          </div>
7.          <div class="col-md-8">
8.              <h1>
9.                  Calculez votre impôt
10.             </h1>
11.         </div>
12.     </div>
13. </div>
```

Commentaires

- lignes 2-13 : le bandeau est encapsulé dans une section Bootstrap de type Jumbotron **[ ]**. Cette classe Bootstrap stylise d'une façon particulière le contenu affiché pour le faire ressortir ;
- lignes 3-12 : une ligne Bootstrap ;
- lignes 4-6 : une image **[img]** est placée dans les quatre premières colonnes de la ligne ;

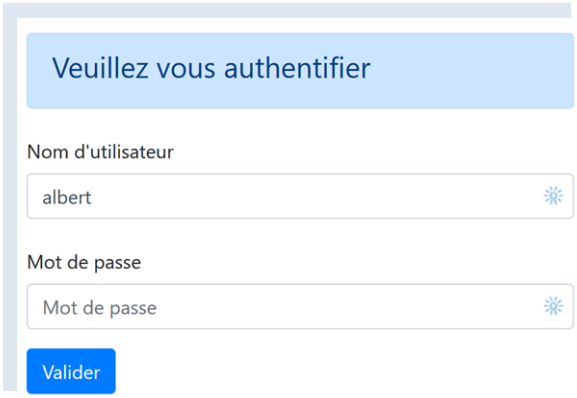
- ligne 5 : la syntaxe **[<?= $logo ?>]** est équivalente à la syntaxe **[<?php print $logo ?>]**. Dit autrement la valeur de l'attribut **[src]** sera la valeur de la variable PHP **[$logo]** ;
- lignes 7-11 : les 8 autres colonnes de la ligne (on rappelle qu'il y en a 12 au total) serviront à placer un texte (ligne 9) en gros caractères (<h1>, lignes 8-10) ;

Eléments dynamiques :

- **[$logo]** : URL de l'image affichée dans le bandeau ;

1.23.13.2.3 Le fragment [v-authentification.php]

Le fragment **[v-authentification .php]** affiche le formulaire d'authentification de l'application web :

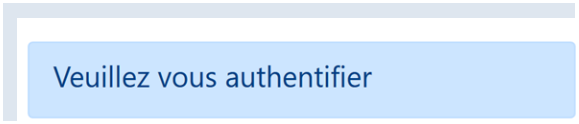


Le code du fragment **[v-authentification.php]** est le suivant :

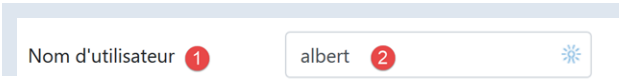
```
1.  <!-- formulaire HTML - on poste ses valeurs avec l'action [authentifier-utilisateur] -->
2.  <form method="post" action="main.php?action=authentifier-utilisateur">
3.
4.      <!-- titre -->
5.      <div class="alert alert-primary" role="alert">
6.          <h4>Veuillez vous authentifier</h4>
7.      </div>
8.
9.      <!-- formulaire Bootstrap -->
10.     <fieldset class="form-group">
11.         <!-- 1re ligne -->
12.         <div class="form-group row">
13.             <!-- libellé -->
14.             <label for="user" class="col-md-3 col-form-label">Nom d'utilisateur</label>
15.             <div class="col-md-4">
16.                 <!-- zone de saisie texte -->
17.                 <input type="text" class="form-control" id="user" name="user"
18.                        placeholder="Nom d'utilisateur" value="<?= $modèle->login ?>">
19.             </div>
20.         </div>
21.         <!-- 2e ligne -->
22.         <div class="form-group row">
23.             <!-- libellé -->
24.             <label for="password" class="col-md-3 col-form-label">Mot de passe</label>
25.             <!-- zone de saisie texte -->
26.             <div class="col-md-4">
27.                 <input type="password" class="form-control" id="password" name="password"
28.                        placeholder="Mot de passe">
29.             </div>
30.         </div>
31.         <!-- bouton de type [submit] sur une 3e ligne-->
32.         <div class="form-group row">
33.             <div class="col-md-2">
34.                 <button type="submit" class="btn btn-primary">Valider</button>
35.             </div>
36.         </div>
37.     </fieldset>
38.
39. </form>
```

Commentaires

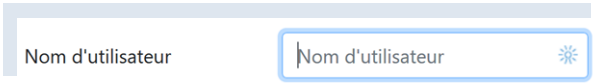
- lignes 2-39 : la balise <form> délimite un formulaire HTML. Celui-ci a en général les caractéristiques suivantes :
  - il définit des zones de saisies (balises <**input**> des lignes 17 et 27 ;
  - il a un bouton de type **[submit]** (ligne 34) qui envoie les valeurs saisies à l'URL indiqué dans l'attribut **[action]** de la balise **[form]** (ligne 2). La méthode HTTP utilisée pour requêter cette URL est précisée dans l'attribut **[method]** de la balise **[form]** (ligne 2) ;
  - ici, lorsque l'utilisateur va cliquer sur le bouton **[Valider]** (ligne 34), le navigateur va poster (ligne 2) les valeurs saisies dans le formulaire à l'URL **[main.php?action=authentifier-utilisateur]** (ligne 2) ;
  - les valeurs postées sont les valeurs saisies par l'utilisateur dans les zones de saisie des lignes 17 et 27. Elles seront postées sous la forme **[user=xx&password=yy]**. Les noms des paramètres **[user, password]** sont ceux des attributs **[name]** des zones de saisie des lignes 17 et 27 ;
- ligne 5-7 : une section Bootstrap pour afficher un titre dans un fond bleu :



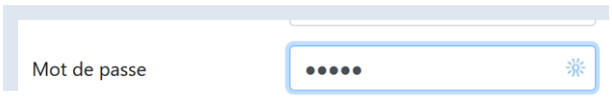
- lignes 10-37 : un formulaire Bootstrap. Tous les éléments du formulaire vont alors être stylisés d'une certaine façon ;
- lignes 12-20 : définissent la 1re ligne du formulaire :



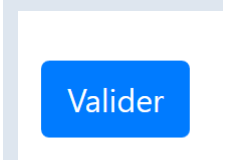
- la ligne 14 définit le libellé **[1]** sur trois colonnes. L'attribut **[for]** de la balise **[label]** relie le libellé à l'attribut **[id]** de la zone de saisie de la ligne 17 ;
- lignes 15-19 : met la zone de saisie dans un ensemble de quatre colonnes ;
- ligne 17 : la balise HTML **[input]** décrit une zone de saisie. Elle a plusieurs paramètres :
  - **[type='text']** : c'est une zone de saisie texte. On peut y taper n'importe quoi ;
  - **[class='form-control']** : style Bootstrap pour la zone de saisie ;
  - **[id='user']** : identifiant de la zone de saisie. Cet identifiant est en général utilisé par le CSS et le code Javascript ;
  - **[name='user']** : nom de la zone de saisie. C'est sous ce nom que la valeur saisie par l'utilisateur sera postée par le navigateur **[user=xx]** ;
  - **[placeholder='invite']** : le texte affiché dans la zone de saisie lorsque l'utilisateur n'a encore rien tapé ;



- **[value='valeur']** : le texte 'valeur' sera affiché dans la zone de saisie dès que celle-ci sera affichée, avant donc que l'utilisateur ne saisisse autre chose. Ce mécanisme est utilisé en cas d'erreur pour afficher la saisie qui a provoqué l'erreur. Ici cette valeur sera la valeur de la variable PHP **[$modèle→login]** ;
- lignes 21-30 : un code analogue pour la saisie du mot de passe ;
- ligne 27 : **[type='password']** fait qu'on a une zone de saisie texte (on peut taper n'importe quoi) mais les caractères tapés sont cachés :



- lignes 32-36 : une 3e ligne pour le bouton **[Valider]** ;
- ligne 34 : parce qu'il a l'attribut **[type=submit]**, un clic sur ce bouton déclenche l'envoi au serveur par le navigateur des valeurs saisies comme il a été expliqué précédemment. L'attribut CSS **[class="btn btn-primary"]** affiche un bouton bleu :



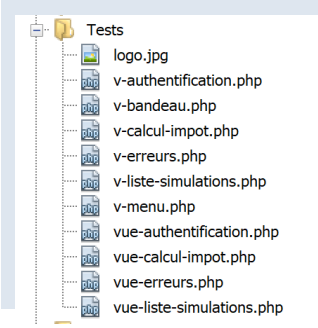
Il nous reste à expliquer une dernière chose. Ligne 2, l'attribut **[action="main.php?action=authentifier-utilisateur"]** définit une URL incomplète (elle ne commence pas par http://machine:port/chemin). Dans notre exemple, toutes les URL de l'application sont de la forme **[http://localhost/php7/scripts-web/impots/version-12/main.php?action=xx]**. La vue de l'authentification va être obtenue avec diverses URL :

- **[http://localhost/php7/scripts-web/impots/version-12/main.php?action=init-session&type=html]**;
- **[http://localhost/php7/scripts-web/impots/version-12/main.php?action=authentifier-utilisateur]**

Ces URL désignent un document **[main.php]** dans le chemin **[http://localhost/php7/scripts-web/impots/version-12]**. Ce sera le cas de toutes les URL de cette application. Le paramètre **[action="main.php?action=authentifier-utilisateur"]** sera préfixé par ce chemin au moment de l'envoi des valeurs saisies. Celles-ci seront donc postées à l'URL **[http://localhost/php7/scripts-web/impots/version-12/main.php?action=authentifier-utilisateur]**.

1.23.13.2.4 Tests visuels

On peut procéder aux tests des vues bien avant leur intégration dans l'application. Il s'agit ici de tester leur aspect visuel. Nous rassemblerons toutes les vues de test dans le dossier **[Tests]** du projet :



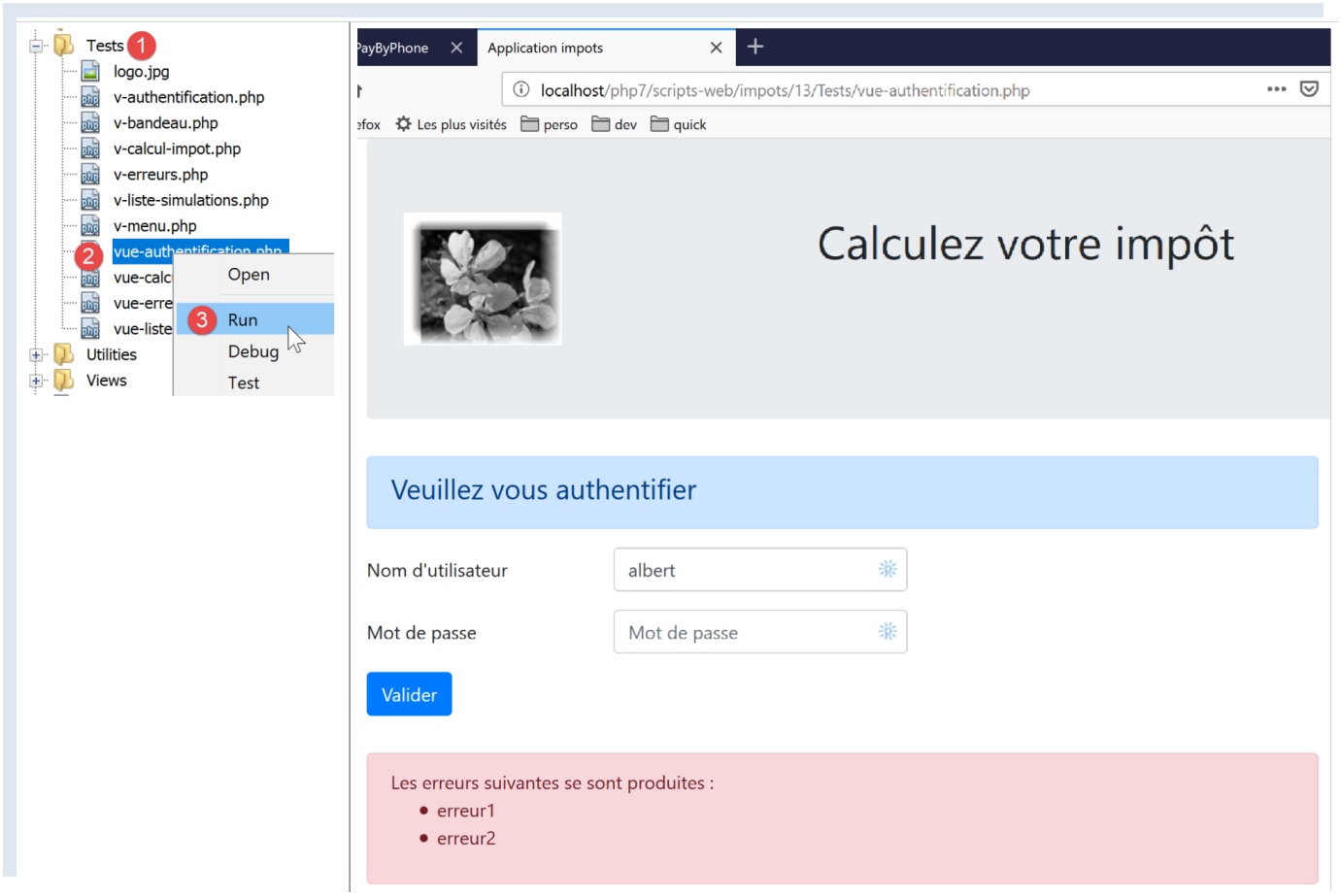
Pour tester la vue **[vue-authentification.php]**, il nous faut créer le modèle de données qu'elle va afficher :

```
1.  <?php
2.  // données de test de la page
3.  //
4.  // on calcule le modèle de la vue
5.  $modèle = getModelForThisView();
6.
7.  function getModelForThisView(): object {
8.    // on encapsule les données de la pagé dans $modèle
9.    $modèle = new \stdClass();
10.   // identifiant utilisateur
11.   $modèle->login = "albert";
12.   // liste d'erreurs
13.   $modèle->error = TRUE;
14.   $erreurs = ["erreur1", "erreur2"];
15.   // on construit une liste HTML des erreurs
16.   $content = "";
17.   foreach ($erreurs as $erreur) {
18.     $content .= "<li>$erreur</li>";
19.   }
20.   $modèle->erreurs = $content;
21.   // image du bandeau
22.   $modèle->logo = "http://localhost/php7/scripts-web/impots/version-12/Tests/logo.jpg";
23.   // on rend le modèle
24.   return $modèle;
25. }
26. ?>
27.
28. <!-- document HTML -->
29. <!doctype html>
30. <html lang="fr">
31.     <head>
32.         <!-- Required meta tags -->
33.         …
34.     </head>
35.     <body>
36.         ….
37.     </body>
38. </html>
```

Commentaires

- lignes 1-5 : la vue d'authentification a des parties dynamiques contrôlées par l'objet **[$modèle]**. On appelle cet objet le **modèle de la vue**. Selon l'une des deux définitions données pour le sigle MVC, on a là le M du MVC ;
- ligne 5 : le modèle de la vue est calculé par la fonction **[getModelForThisView]** ;
- ligne 9 : le modèle de la vue sera encapsulé dans un type **[stdClass]** ;
- lignes 10-22 : on définit des valeurs de tests pour les éléments dynamiques de la vue d'authentification ;

Le test visuel peut se faire à partir de Netbeans :



On poursuit ces tests visuels jusqu'à être satisfait du résultat.

1.23.13.2.5 Calcul du modèle de la vue

Une fois l'aspect visuel de la vue déterminé, on peut procéder au calcul du modèle de la vue en conditions réelles. Rappelons les codes d'état qui mènent à cette vue. On les trouve dans le fichier de configuration :

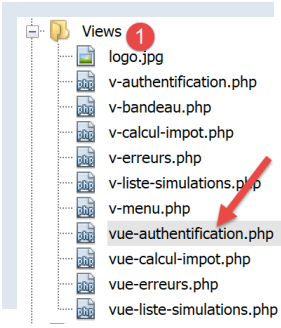
```
1.  "vues": {
2.          "vue-authentification.php": [700, 221, 400],
3.          "vue-calcul-impot.php": [200, 300, 341, 350, 800],
4.          "vue-liste-simulations.php": [500, 600]
5.      },
6.  "vue-erreurs": "vue-erreurs.php"
```

Ce sont donc les codes d'état **[700, 221, 400]** qui font afficher la vue d'authentifcation. Pour retrouver la signification de ces codes, on peut s'aider des tests **[Postman]** réalisés sur l'application jSON :

- **[init-session-json-700]** : 700 est le code d'état à l'issue d'une action **[init-session]** réussie : on présente alors le formulaire d'authentification vide ;
- **[authentifier-utilisateur-221]** : 221 est le code d'état à l'issue d'une action **[authentifier-utilisateur]** ayant échoué (identifiants non reconnus) : on présente alors le formulaire d'authentification pour qu'il soit corrigé ;

- **[fin-session-400]** : 400 est le code d'état à l'issue d'une action **[fin-session]** réussie : on présente alors le formulaire d'authentification vide ;

Maintenant que nous savons à quels moments doit être affiché le formulaire d'authentification, on peut calculer son modèle dans **[vue-authentification.php]** :



Le code de calcul du modèle de la vue **[vue-authentification.php]** est le suivant :

```
1.  <?php
2.  // on hérite des variables suivantes
3.  // Request $request : la requête en cours
4.  // Session $session : la session de l'application
5.  // array $config : la configuration de l'application
6.  // array $content : la réponse du contrôleur
7.  //
8.  // dépendances Symfony
9.  use Symfony\Component\HttpFoundation\Request;
10. use Symfony\Component\HttpFoundation\Session\Session;
11.
12. // on calcule le modèle de la vue
13. $modèle = getModelForThisView($request, $session, $config, $content);
14.
15. function getModelForThisView(Request $request, Session $session, array $config, array $content): object {
16.   // on encapsule les données de la pagé dans $modèle
17.   $modèle = new stdClass();
18.   // état de l'application
19.   $état = $content["état"];
20.   // le modèle dépend de l'état
21.   switch ($état) {
22.     case 700:
23.     case 400:
24.       // cas de l'affichage du formulaire vide
25.       $modèle->login = "";
26.       // il n'y pas d'erreur à afficher
27.       $modèle->error = FALSE;
28.       break;
29.     case 221:
30.       // authentification erronée
31.       // on réaffiche l'utilisateur initialement saisi
32.       $modèle->login = $request->request->get("user");
33.       // il y a une erreur à afficher
34.       $modèle->error = TRUE;
35.       // liste HTML de msg d'erreur - ici un seul
36.       $modèle->erreurs = "<li>Echec de l'authentification</li>";
37.   }
38.   // résultat
39.   return $modèle;
40. }
41. ?>
42.
43. <!-- document HTML -->
44. <!doctype html>
45. <html lang="fr">
46.     <head>
47.         …
48.     </head>
49.     <body>
50.         …
51.     </body>
52. </html>
```

Commentaires

- lignes 3-6 : on rappelle les variables héritées de la classe **[HtmlResponse]** qui fait afficher par un **[require]** la vue **[vue-authentification.php]** ;
- lignes 9-10 : les classes Symfony utilisées dans le code de la vue ;
- lignes 15-40 : la fonction **[getModelForThisView]** est chargée de calculer le modèle de la vue ;
- ligne 19 : on récupère le code d'état rendu par le contrôleur qui a traité l'action en cours ;
- lignes 21-37 : le modèle dépend de ce code d'état ;
- lignes 22-28 : cas où on doit afficher un formulaire d'authentification vierge ;
- lignes 29-37 : cas de l'authentification erronée : on raffiche l'identifiant saisi pour l'utilisateur et on affiche un message d'erreur. L'utilisateur au clavier peut alors réessayer une autre authentification ;

Un modèle particulier a été écrit pour le bandeau **[v-bandeau.php]** :

```
1.  <?php
2.    // logo
3.    $scheme = $request->server->get('REQUEST_SCHEME'); // http
4.    $host = $request->server->get('SERVER_NAME'); // localhost
5.    $port = $request->server->get('SERVER_PORT'); // 80
6.    $uri = $request->server->get('REQUEST_URI'); // /php7/scripts-web/impots/version-12/main.php?action=xxx
7.    $champs = [];
8.    preg_match("/(.+)\/.+?$/", $uri, $champs);
9.    $root = $champs[1]; // /php7/scripts-web/impots/version-12
10.   $modèle->logo = "$scheme://$host:$port$root/Views/logo.jpg"; // http://localhost:80/php7/scripts-
    web/impots/version-12/Views/logo.jpg
11. ?>
12. <!-- Bootstrap Jumbotron -->
13. <div class="jumbotron">
14.     <div class="row">
15.         <div class="col-md-4">
16.             <img src="<?= $modèle->logo ?>" alt="Cerisier en fleurs" />
17.         </div>
18.         <div class="col-md-8">
19.             <h1>
20.                 Calculez votre impôt
21.             </h1>
22.         </div>
23.     </div>
24. </div>
```

Commentaires

- la ligne 16 utilise la variable **[$modèle→logo]** qui est l'URL du logo du bandeau. Plutôt que de calculer cette variable quatre fois pour les quatre vues de l'application, ce calcul est factorisé dans le fragment **[v-bandeau.php]** ;
- les lignes 1-11 montrent comment construire l'URL **[http://localhost:80/php7/scripts-web/impots/version-12/Views/logo.jpg]** à partir des informations trouvées dans l'environnement du serveur **[$request→server]** ;

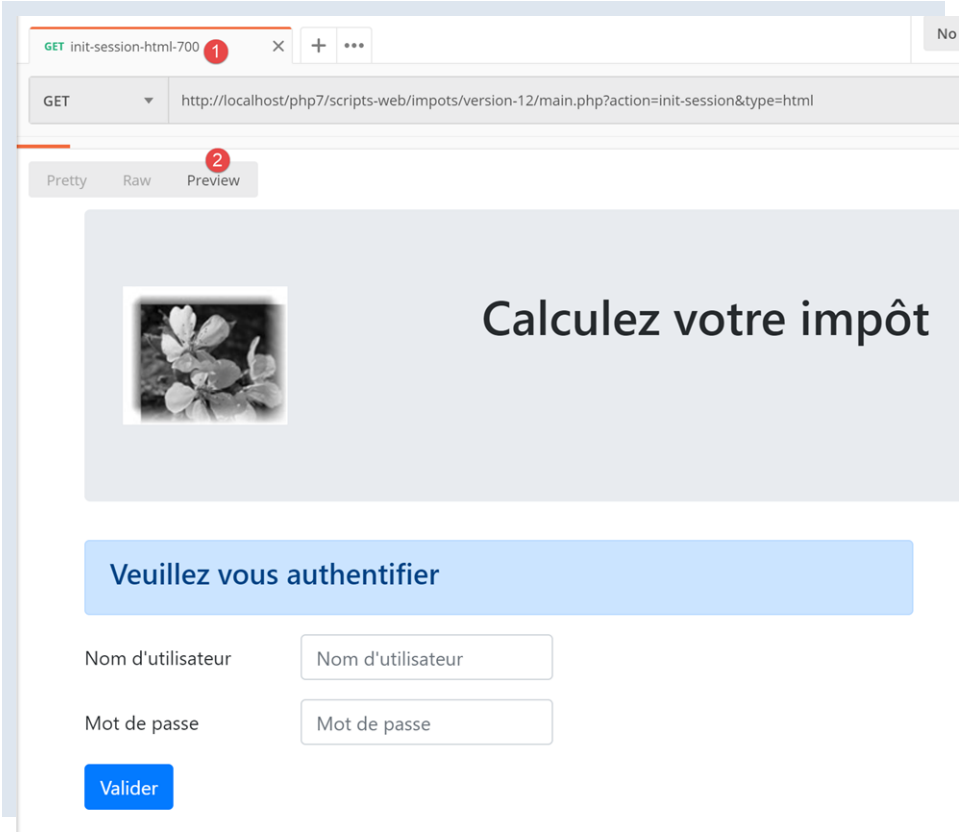
1.23.13.2.6 Tests [Postman]

Nous avons déjà créé des requêtes produidant les codes **[700, 221, 400]** qui affichent la vue d'authentification. Rappelons-les :

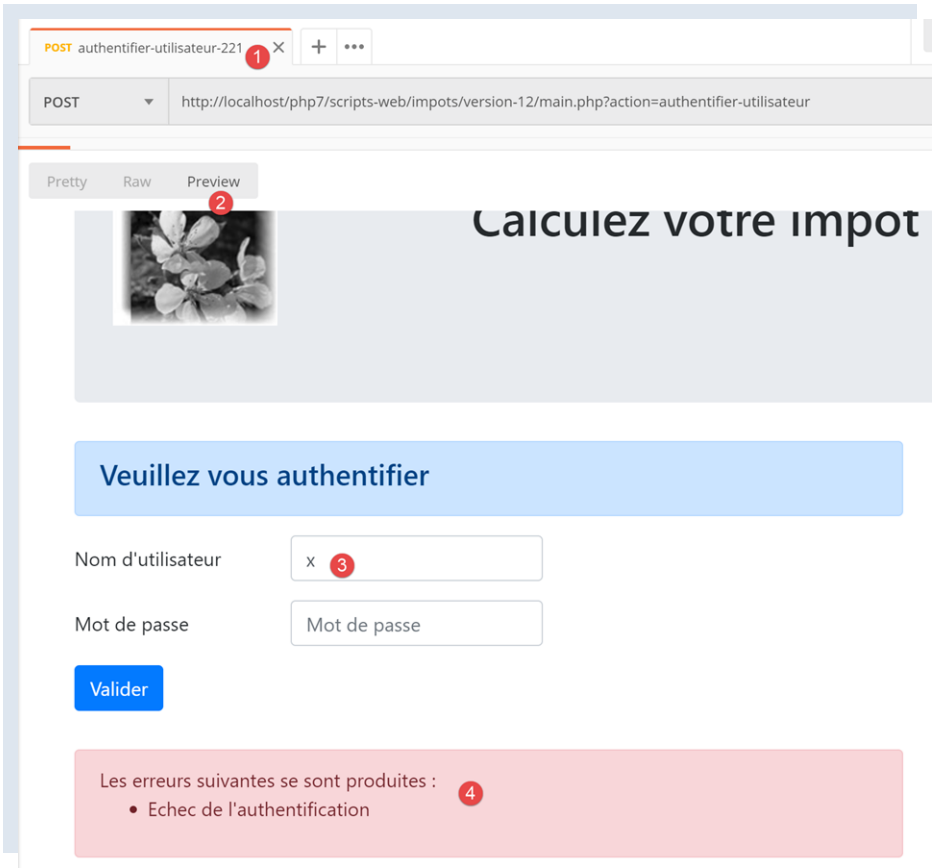
- **[init-session-html-700]** : 700 est le code d'état à l'issue d'une action **[init-session]** réussie : on présente alors le formulaire d'authentification vide ;
- **[authentifier-utilisateur-221]** : 221 est le code d'état à l'issue d'une action **[authentifier-utilisateur]** ayant échoué (identifiants non reconnus) : on présente alors le formulaire d'authentification pour qu'il soit corrigé ;
- **[fin-session-400]** : 400 est le code d'état à l'issue d'une action **[fin-session]** réussie : on présente alors le formulaire d'authentification vide ;

Il suffit de les réutiliser et de voir s'ils affichent bien la vue d'authentification. On ne montrera ici que deux tests :

- **[init-session-html-700]** : début d'une session HTML ;



- **`[authentifier-utilisateur-221]`** : authentification de l'utilisateur **`[x, x]`** ;



Ci-dessus :

- la requête avait posté la chaîne **[user=x&password=x]** ;
- en **[4]**, un message d'erreur est affiché ;
- en **[3]**, l'utilisateur erroné a été réaffiché ;

1.23.13.2.7 Conclusion

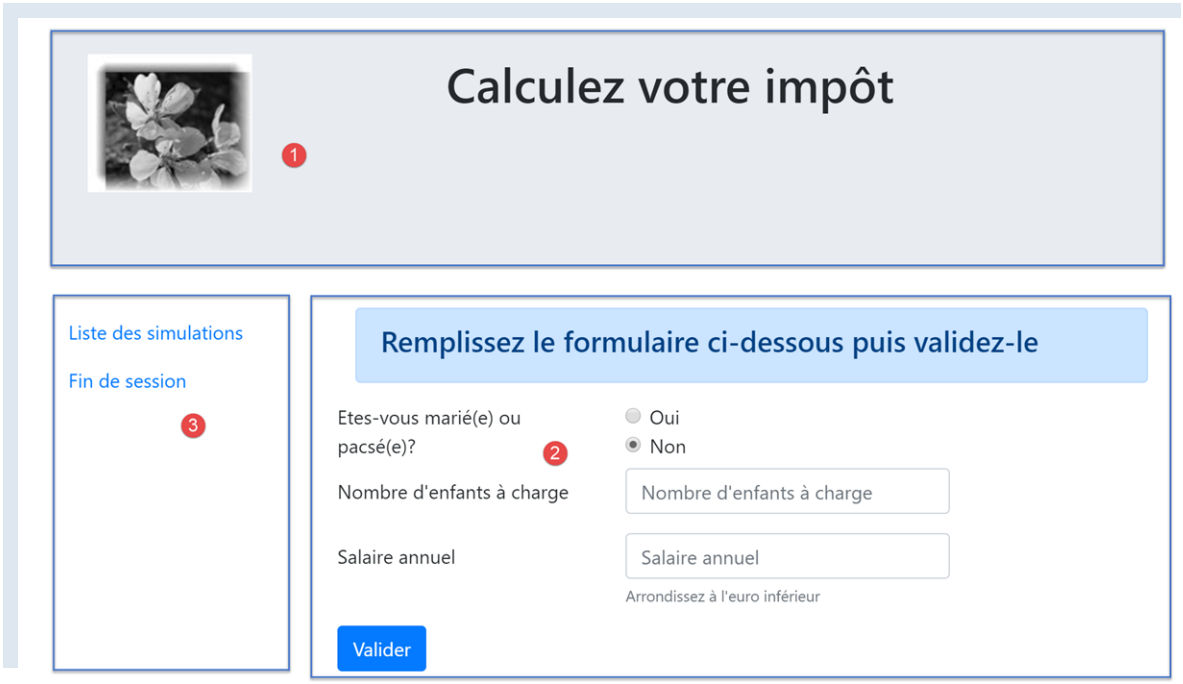
Nous avons pu tester la vue **[vue-authentification.php]** sans avoir écrit les autres vues. Cela a été possible parce que :

- tous les contrôleurs sont écrits ;
- **[Postman]** nous permet d'émettre des requêtes vers le serveur sans avoir besoin des vues. Lorsqu'on écrit les contrôleurs, il faut être conscient que n'importe qui peut faire cela. Il faut donc être prêt à traiter des requêtes qu'aucune vue ne permettrait. Elles sont fabriquées à la main dans **[Postman]**. On ne doit jamais se dire a priori « cette requête est impossible » . Il faut vérifier ;

1.23.13.3 La vue de calcul de l'impôt

1.23.13.3.1 Présentation de la vue

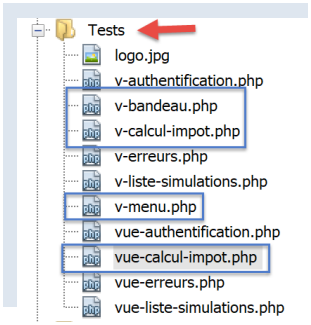
La vue de calcul de l'impôt est la suivante :



La vue a trois parties :

- 1 : le bandeau supérieur est généré par le fragment **[v-bandeau.php]** déjà présenté ;
- 2 : le formulaire de calcul de l'impôt généré par le fragment **[v-calcul-impot.php]** ;
- 3 : un menu présentant deux liens, généré par le fragment **[v-menu.php]** ;

La vue de calcul de l'impôt est générée par le script **[vue-calcul-impot.php]** suivant :



```php
<?php
// on hérite des variables suivantes
// Request $request : la requête en cours
// Session $session : la session de l'application
// array $config : la configuration de l'application
// array $content : la réponse du contrôleur qui a traité l'action
//
// dépendances Symfony
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Session\Session;

// on calcule le modèle de la vue
$modèle = getModelForThisView($request, $session, $config, $content);

function getModelForThisView(Request $request, Session $session, array $config, array $content): object {
  // on encapsule les données de la pagé dans $modèle
  $modèle = new \stdClass();
  …
  // on rend le modèle
  return $modèle;
}
?>
<!-- document HTML -->
<!doctype html>
<html lang="fr">
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <!-- Bootstrap CSS -->
        <link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-
    MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
        <title>Application impots</title>
    </head>
    <body>
        <div class="container">
            <!-- bandeau -->
            <?php require "v-bandeau.php"; ?>
            <!-- ligne à deux colonnes -->
            <div class="row">
                <!-- le menu -->
                <div class="col-md-3">
                    <?php require "v-menu.php" ?>
                </div>
                <!-- le formulaire de calcul -->
                <div class="col-md-9">
                    <?php require "v-calcul-impot.php" ?>
                </div>
            </div>
            <!-- cas du succès -->
            <?php
            if ($modèle->success) {
              // on affiche une alerte de réussite
              print <<<EOT1
            <div class="row">
                <div class="col-md-3">

                </div>
                <div class="col-md-9">
                    <div class="alert alert-success" role="alert">
                        $modèle->impôt</br>
                        $modèle->décôte</br>\n
                        $modèle->réduction</br>\n
                        $modèle->surcôte</br>\n
                        $modèle->taux</br>\n
                    </div>
                </div>
            </div>
EOT1;
            }
            ?>
            <?php
            if ($modèle->error) {
              // liste des erreurs sur 9 colonnes
              print <<<EOT2
                <div class="row">
```

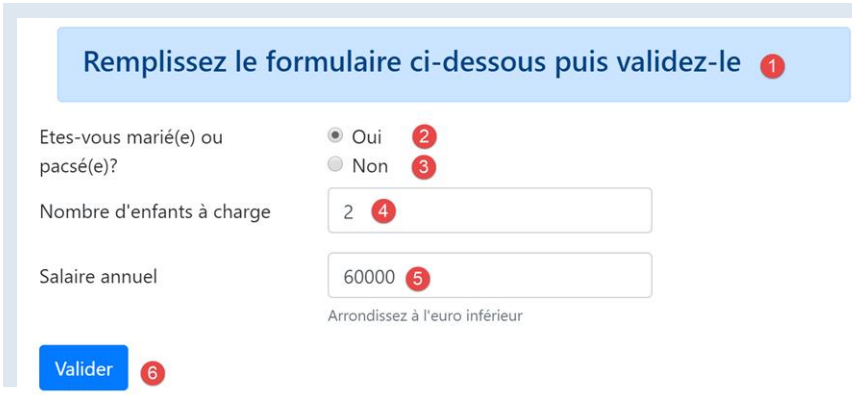
```
76.                     <div class="col-md-3">
77.
78.                     </div>
79.                     <div class="col-md-9">
80.                         <div class="alert alert-danger" role="alert">
81.                           L'erreur suivante s'est produite :
82.                           <ul>$modèle->erreurs</ul>
83.                         </div>
84.                     </div>
85.                 </div>
86. EOT2;
87.             }
88.             ?>
89.         </div>
90.     </body>
91. </html>
```

Commentaires

- nous ne commentons que des nouveautés encore non rencontrées ;
- ligne 37 : inclusion du bandeau supérieur de la vue dans la première ligne Bootstrap de la vue ;
- lignes 41-43 : inclusion du menu qui occupera trois colonnes de la seconde ligne Bootstrap de la vue ;
- lignes 45-47 : inclusion du formulaire de calcul d'impôt qui occupera neuf colonnes de la seconde ligne Bootstrap de la vue ;
- lignes 51-69 : si le calcul de l'impôt réussit **`[$modèle→success=TRUE]`**, alors le résultat du calcul de l'impôt est affiché dans un cadre vert (lignes 59-65). Ce cadre est dans la troisième ligne Bootstrap de la vue (ligne 54) et occupe neuf colonnes (ligne 58) à droite de trois colonnes vides (lignes 55-57). Ce cadre sera donc immédiatement dessous le formulaire de calcul de l'impôt ;
- lignes 71-87 : si le calcul de l'impôt échoue **`[$modèle→error=TRUE]`**, alors un message d'erreur est affiché dans un cadre rose (lignes 80-83). Ce cadre est dans la troisième ligne Bootstrap de la vue (ligne 75) et occupe neuf colonnes (ligne 79) à droite de trois colonnes vides (lignes 76-78). Ce cadre sera donc immédiatement dessous le formulaire de calcul de l'impôt ;

1.23.13.3.2 Le fragment `[v-calcul-impot.php]`

Le fragment **`[v-calcul-impot.php]`** affiche le formulaire d'authentification de l'application web :



Le code du fragment **`[v-calcul-impot.php]`** est le suivant :

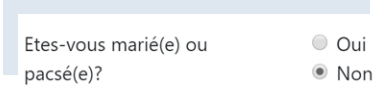
```
1.  <!-- formulaire HTML posté -->
2.  <form method="post" action="main.php?action=calculer-impot">
3.      <!-- message sur 12 colonnes sur fond bleu -->
4.      <div class="col-md-12">
5.          <div class="alert alert-primary" role="alert">
6.              <h4>Remplissez le formulaire ci-dessous puis validez-le</h4>
7.          </div>
8.      </div>
9.      <!-- éléments du formulaire -->
10.     <fieldset class="form-group">
11.         <!-- première ligne sur 9 colonnes -->
12.         <div class="row">
13.             <!-- libellé sur 4 colonnes -->
14.             <legend class="col-form-label col-md-4 pt-0">Etes-vous marié(e) ou pacsé(e)?</legend>
15.             <!-- boutons radio sur 5 colonnes-->
16.             <div class="col-md-5">
17.                 <div class="form-check">
18.                     <input class="form-check-input" type="radio" name="marié" id="gridRadios1" value="oui"
    <?= $modèle->checkedOui ?>>
19.                     <label class="form-check-label" for="gridRadios1">
20.                         Oui
```

```
21.                     </label>
22.                 </div>
23.                 <div class="form-check">
24.                     <input class="form-check-input" type="radio" name="marié" id="gridRadios2" value="non"
    <?= $modèle->checkedNon ?>>
25.                     <label class="form-check-label" for="gridRadios2">
26.                         Non
27.                     </label>
28.                 </div>
29.             </div>
30.         </div>
31.         <!-- deuxième ligne sur 9 colonnes -->
32.         <div class="form-group row">
33.             <!-- libellé sur 4 colonnes -->
34.             <label for="enfants" class="col-md-4 col-form-label">Nombre d'enfants à charge</label>
35.             <!-- zone de saisie numérique du nombre d'enfants sur 5 colonnes -->
36.             <div class="col-md-5">
37.                 <input type="number" min="0" step="1" class="form-control" id="enfants" name="enfants"
    placeholder="Nombre d'enfants à charge" value="<?= $modèle->enfants ?>">
38.             </div>
39.         </div>
40.         <!-- troisième ligne sur 9 colonnes -->
41.         <div class="form-group row">
42.             <!-- libellé sur 4 colonnes -->
43.             <label for="salaire" class="col-md-4 col-form-label">Salaire annuel</label>
44.             <!-- zone de saisie numérique pour le salaire sur 5 colonnes -->
45.             <div class="col-md-5">
46.                 <input type="number" min="0" step="1" class="form-control" id="salaire" name="salaire"
    placeholder="Salaire annuel" aria-describedby="salaireHelp" value="<?= $modèle->salaire ?>">
47.                 <small id="salaireHelp" class="form-text text-muted">Arrondissez à l'euro inférieur</small>
48.             </div>
49.         </div>
50.         <!-- quatrième ligne, bouton [submit] sur 5 colonnes -->
51.         <div class="form-group row">
52.             <div class="col-md-5">
53.                 <button type="submit" class="btn btn-primary">Valider</button>
54.             </div>
55.         </div>
56.     </fieldset>
57.
58. </form>
```

Commentaires

- ligne 2 : le formulaire HTML sera posté (attribut **[method]**) à l'URL **[main.php?action=calculer-impot]** (attribut **[action]**). Les valeurs postées seront les valeurs des zones de saisie :
  - la valeur du bouton radio coché sous la forme :
    - **[marié=oui]** si le bouton radio **[Oui]** est coché (lignes 16-22). **[marié]** est la valeur de l'attribut **[name]** de la ligne 18, **[oui]** la valeur de l'attribut **[value]** de la ligne 18 ;
    - **[marié=non]** si le bouton radio **[Non]** est coché (lignes 23-28). **[marié]** est la valeur de l'attribut **[name]** de la ligne 24, **[non]** la valeur de l'attribut **[value]** de la ligne 24 ;
  - la valeur de la zone de saisie numérique de la ligne 37 sous la forme **[enfants=xx]** où **[enfants]** est la valeur de l'attribut **[name]** de la ligne 37, et **[xx]** la valeur saisie par l'utilisateur au clavier ;
  - la valeur de la zone de saisie numérique de la ligne 46 sous la forme **[salaire=xx]** où **[salaire]** est la valeur de l'attribut **[name]** de la ligne 46, et **[xx]** la valeur saisie par l'utilisateur au clavier ;

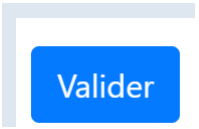
  Finalement, la valeur postée aura la forme **[marié=xx&enfants=yy&salaire=zz]**.
- les valeurs saisies seront postées lorsque l'utilisateur cliquera sur le bouton de type **[submit]** de la ligne 53 ;
- lignes 16-30 : les deux boutons radio :



  Les deux boutons radio font partie du même groupe de boutons radio car ils ont le même attribut **[name]** (lignes 18, 24). Le navigateur s'assure que dans un groupe de boutons radio, un seul est coché à un moment donné. Donc cliquer sur l'un désactive celui qui était coché auparavant ;
- ce sont des boutons radio à cause de l'attribut **[type="radio"]** (lignes 18, 24) ;
- à l'affichage du formulaire (avant saisie), l'un des boutons radio devra être coché : il suffit pour cela d'ajouter l'attribut **[checked='checked']** à la balise <input **type="radio"**> concernée. Cela est réalisé avec des variables dynamiques :
  - **[<?= $modèle->checkedOui ?>]** à la ligne 18 ;

    - `[<?= $modèle->checkedNon ?>]` à la ligne 24 ;
      - Ces variables feront partie du modèle de la vue.
- ligne 37 : une zone de saisie numérique `[type="number"]` avec une valeur minimale de 0 `[min="0"]`. Dans les navigateurs récents, cela signifie que l'utilisateur ne pourra saisir qu'un nombre >=0. Sur ces mêmes navigateurs récents, la saisie peut être faite avec un variateur qu'on peut cliquer à la hausse ou à la baisse. L'attribut `[step="1"]` de la ligne 37 indique que le variateur opèrera avec des sauts de 1 unité. Cela a pour conséquence que le variateur ne prendra pour valeur que des entiers progressant de 0 à n avec un pas de 1. Pour la saisie manuelle, cela signifie que les nombres avec virgule ne seront pas acceptés ;

- ligne 37 : lors de certains affichages, la zone de saisie des enfants devra être préremplie avec la dernière saisie faite dans cette zone. On utilise pour cela l'attribut `[value]` qui fixe la valeur à afficher dans la zone de saisie. Cette valeur sera dynamique et générée par la variable `[$modèle→enfants]` ;
- ligne 46 : mêmes explications pour la saisie du salaire que pour celle des enfants ;
- ligne 53 : le bouton de type `[submit]` qui déclenche le POST des valeurs saisies à l'URL `[main.php?action=calculer-impot]` ;



1.23.13.3.3 Le fragment `[v-menu.php]`

Ce fragment affiche un menu à gauche du formulaire de calcul de l'impôt :

Le code de ce fragment est le suivant :

```
1.  <!-- menu Bootstrap -->
2.  <nav class="nav flex-column">
3.      <?php
4.      // affichage d'une liste de liens HTML
5.      foreach($modèle->optionsMenu as $texte=>$url){
6.        print <<<EOT3
7.        <a class="nav-link" href="$url">$texte</a>
8.  EOT3;
9.      }
10.     ?>
11. </nav>
```

Commentaires

- lignes 2-11 : la balise HTML `[nav]` encadre une portion de document HTML présentant des liens de navigation vers d'autres documents ;
- ligne 7 : la balise HTML `[a]` introduit un lien de navigation :
    - `[$url]` : est l'URL vers laquelle on navigue lorsqu'on clique sur le lien `[$texte]`. C'est alors une opération `[GET $url]` qui est faite par le navigateur. Si `[$url]` est une URL relative alors elle est préfixée par la racine de l'URL actuellement affichée dans l'adresse du navigateur. Ainsi pour obtenir le lien `[1]`, alors que l'URL actuelle du navigateur est du type `[http://chemin/main.php?paramètres]`, on créera le lien :

```
<a href='main.php?action=liste-simulation'>Liste des simulations</a>
```

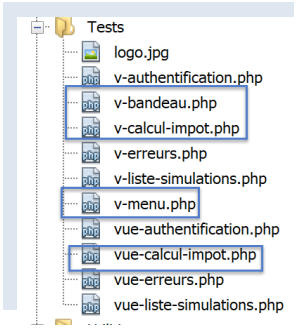
- ligne 5 : le modèle **[$modèle→optionsMenu]** du fragment sera un tableau de la forme :

```
['  Liste des simulations'=>'main.php?action=liste-simulations',
'  Fin de session'=>'main.php?action=fin-session']
```

- lignes 2, 7 : les classes CSS **[nav, flex-column, nav-link]** sont des classes Bootstrap qui donnent son apparence au menu ;

1.23.13.3.4 Test visuel

Nous rassemblons ces différents éléments dans le dossier **[Tests]** et nous créons un modèle de test pour la vue **[vue-calcul-impot.php]** :



Le modèle de données de la vue **[vue-calcul-impot]** sera le suivant :

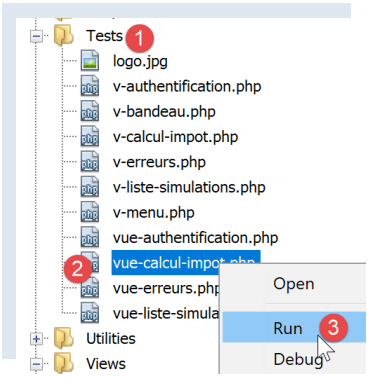
```
1.  <?php
2.  // données de test de la page
3.  //
4.  // on calcule le modèle de la vue
5.  $modèle = getModelForThisView();
6.
7.  function getModelForThisView(): object {
8.    // on encapsule les données de la pagé dans $modèle
9.    $modèle = new \stdClass();
10.   // formulaire
11.   $modèle->checkedOui = "";
12.   $modèle->checkedNon = 'checked="checked"';
13.   $modèle->enfants = 2;
14.   $modèle->salaire = 300000;
15.   // message de réussite
16.   $modèle->success = TRUE;
17.   $modèle->impôt = "Montant de l'impôt : 1000 euros";
18.   $modèle->décôte = "Décôte : 15 euros";
19.   $modèle->réduction = "Réduction : 20 euros";
20.   $modèle->surcôte = "Surcôte : 0 euros";
21.   $modèle->taux = "Taux d'imposition : 14 %";
22.   // message d'erreur
23.   $modèle->error = TRUE;
24.   $erreurs = ["erreur1", "erreur2"];
25.   // on construit une liste HTML des erreurs
26.   $content = "";
27.   foreach ($erreurs as $erreur) {
28.     $content .= "<li>$erreur</li>";
29.   }
30.   $modèle->erreurs = $content;
31.   // menu
32.   $modèle->optionsMenu = [
33.     'Liste des simulations' => 'main.php?action=liste-simulations',
34.     'Fin de session' => 'main.php?action=fin-session'];
35.   // image du bandeau
36.   $modèle->logo = "http://localhost/php7/scripts-web/impots/version-12/Tests/logo.jpg";
37.   // on rend le modèle
38.   return $modèle;
39. }
40.
41. ?>
42. <!-- document HTML -->
43. <!doctype html>
44. <html lang="fr">
45.     <head>
```

```
46.         …
47.     </head>
48.     <body>
49.         …
50.     </body>
51. </html>
```

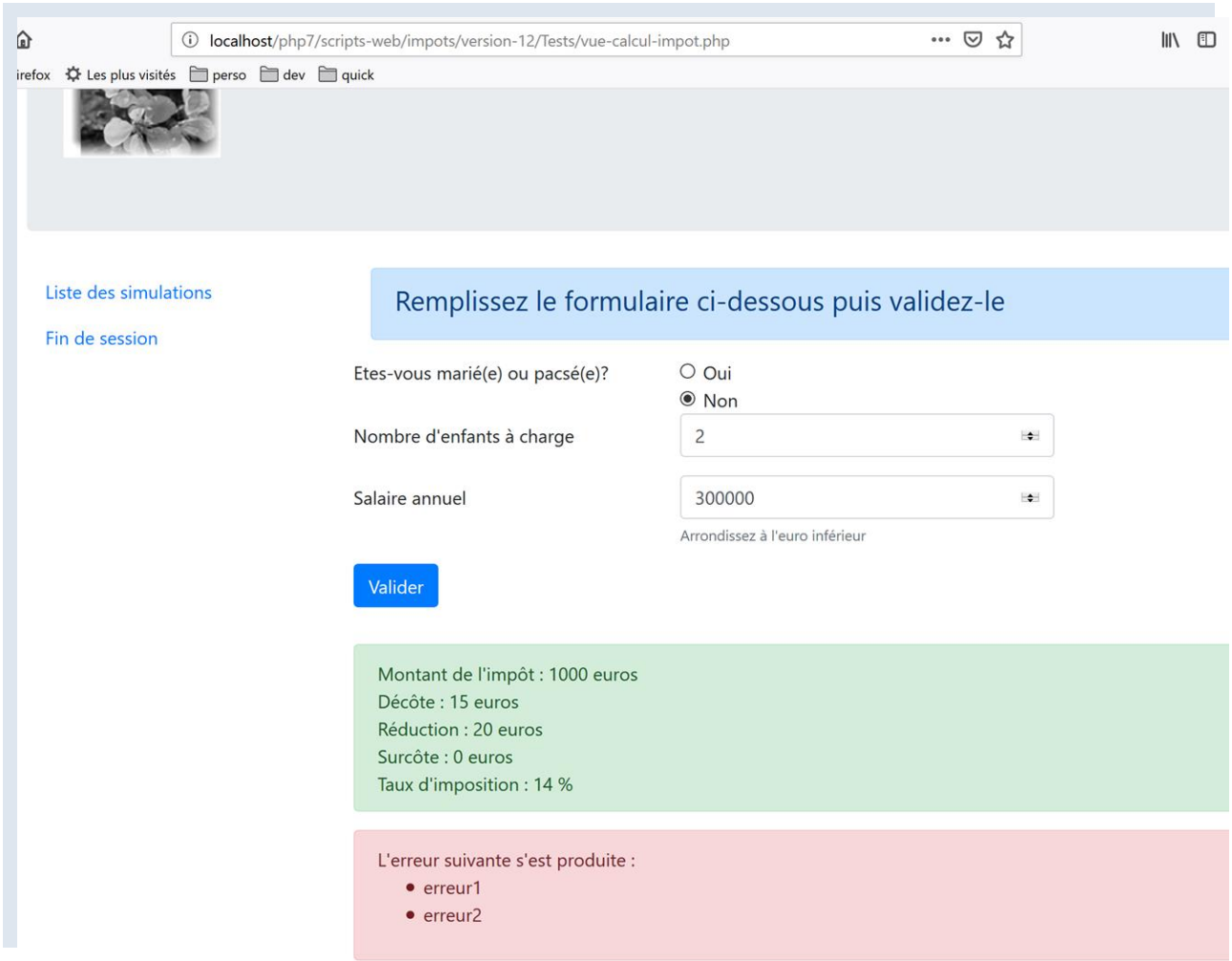
Commentaires

- lignes 7-39 : on initialise toutes les parties dynamiques de la vue **[vue-calcul-impot.php]** et des fragments **[v-calcul-impot.php]** et **[v-menu.php]** ;

On teste la vue **[vue-calcul-impot.php]** :

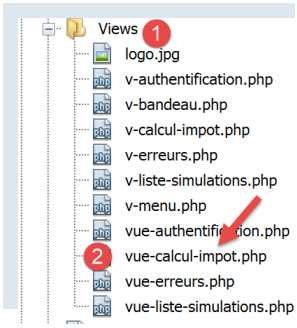


On obtient le résultat suivant :



On travaille sur cette vue jusqu'à ce que le résultat obtenu visuellement nous convienne. On peut ensuite passer à l'intégration de la vue dans l'application web en cours d'écriture.

1.23.13.3.5 Calcul du modèle de la vue



Une fois l'aspect visuel de la vue déterminé, on peut procéder au calcul du modèle de la vue en conditions réelles. Rappelons les codes d'état qui mènent à cette vue. On les trouve dans le fichier de configuration :

```
1.  "vues": {
2.          "vue-authentification.php": [700, 221, 400],
3.          "vue-calcul-impot.php": [200, 300, 341, 350, 800],
4.          "vue-liste-simulations.php": [500, 600]
5.      },
6.  "vue-erreurs": "vue-erreurs.php"
```

Ce sont donc les codes d'état **[200, 300, 341, 350, 800]** qui font afficher la vue d'authentifcation. Pour retrouver la signification de ces codes, on peut s'aider des tests **[Postman]** réalisés sur l'application jSON :

- **[authentifier-utilisateur-200]** : 200 est le code d'état à l'issue d'une action **[authentifier-itilisateur]** réussie : on présente alors le formulaire de calcul d'impôt vide ;
- **[calculer-impot-300]** : 300 est le code d'état à l'issue d'une action **[calculer-impot]** réussie. On affiche alors le formulaire de calcul avec les données qui y ont été saisies et le montant de l'impôt. L'utilisateur peut alors refaire un autre calcul ;
- **[fin-session-400]** : 400 est le code d'état à l'issue d'une action **[fin-session]** réussie : on présente alors le formulaire d'authentification vide ;
- le code d'état **[341]** est celui obtenu pour un calcul d'impôt valide mais l'absence de connexion au SGBD provoque une erreur ;
- le code d'état **[350]** est celui obtenu pour un calcul d'impôt valide mais l'absence de connexion au serveur **[Redis]** provoque une erreur ;
- le codes d'état **[800]** sera présenté ultérieurement. Nous ne l'avons pas encore rencontré ;
- on a fait ici l'hypothèse que l'utilisateur utilise un navigateur récent. Ainsi avec le formulaire étudié, il n'est pas possible de taper des nombres négatifs, des chaines de caractères non numériques, des nombres à virgule dans les champs de saisie **[enfants, salaire]**. Avec des navigateurs plus anciens, ce serait possible. On traitera ces erreurs comme des erreurs inattendues et on affichera alors la vue **[vue-erreurs]** ;

Maintenant que nous savons à quels moments doit être affiché le formulaire de calcul de l'impôt, on peut calculer son modèle dans **[vue-calcul-impot.php]** :

```
1.  <?php
2.  // on hérite des variables suivantes
3.  // Request $request : la requête en cours
4.  // Session $session : la session de l'application
5.  // array $config : la configuration de l'application
6.  // array $content : la réponse du contrôleur qui a traité l'action
7.  //
8.  // dépendances Symfony
9.  use Symfony\Component\HttpFoundation\Request;
10. use Symfony\Component\HttpFoundation\Session\Session;
11.
12. // on calcule le modèle de la vue
13. $modèle = getModelForThisView($request, $session, $config, $content);
14.
15. function getModelForThisView(Request $request, Session $session, array $config, array $content): object {
16.   // on encapsule les données de la pagé dans $modèle
17.   $modèle = new \stdClass();
18.   // état de l'application
19.   $état = $content["état"];
20.   // le modèle dépend de l'état
```

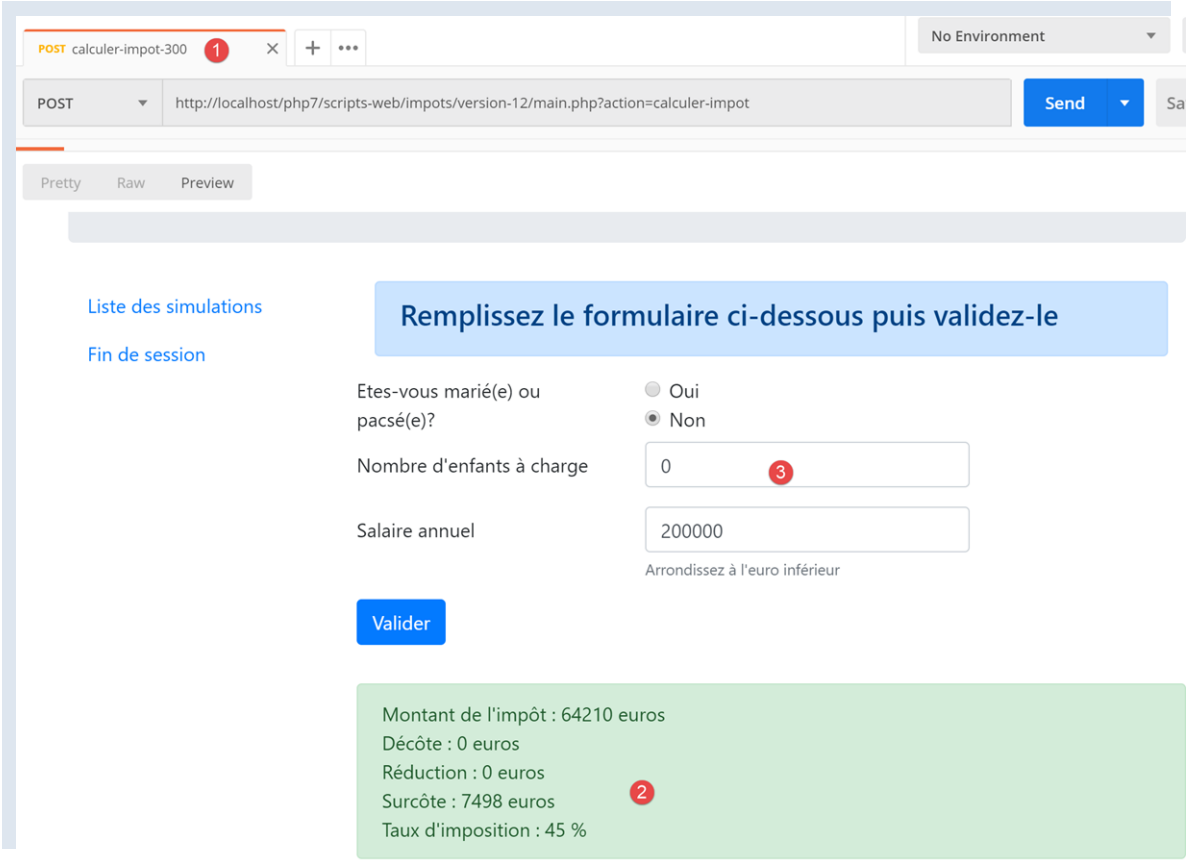
```
21.   switch ($état) {
22.     case 200 :
23.     case 800:
24.       // affichage initial d'un formulaire vide
25.       $modèle->success = FALSE; $modèle->errror = FALSE;
26.       $modèle->checkedNon = 'checked="checked"';
27.       $modèle->checkedOui = "";
28.       $modèle->enfants = "";
29.       $modèle->salaire = "";
30.       break;
31.     case 300:
32.       // réussite du calcul - affichage du résultat
33.       $modèle->success = TRUE;
34.       $modèle->error = FALSE;
35.       $modèle->impôt = "Montant de l'impôt : {$content["réponse"]["impôt"]} euros";
36.       $modèle->décôte = "Décôte : {$content["réponse"]["décôte"]} euros";
37.       $modèle->réduction = "Réduction : {$content["réponse"]["réduction"]} euros";
38.       $modèle->surcôte = "Surcôte : {$content["réponse"]["surcôte"]} euros";
39.       $modèle->taux = "Taux d'imposition : " . ($content["réponse"]["taux"] * 100) . " %";
40.       // formulaire rétabli avec les valeurs saisies
41.       $modèle->checkedOui = $request->request->get("marié") === "oui" ? 'checked="checked"' : "";
42.       $modèle->checkedNon = $request->request->get("marié") === "oui" ? "" : 'checked="checked"';
43.       $modèle->enfants = $request->request->get("enfants");
44.       $modèle->salaire = $request->request->get("salaire");
45.       break;
46.     case 341:
47.     // base de données HS
48.     case 350:
49.       // serveur Redis HS
50.       // formulaire rétabli avec les valeurs saisies
51.       $modèle->checkedOui = $request->request->get("marié") === "oui" ? 'checked="checked"' : "";
52.       $modèle->checkedNon = $request->request->get("marié") === "oui" ? "" : 'checked="checked"';
53.       $modèle->enfants = $request->request->get("enfants");
54.       $modèle->salaire = $request->request->get("salaire");
55.       // erreur
56.       $modèle->success = FALSE;
57.       $modèle->error = TRUE;
58.       $modèle->erreurs = "<li>{$content["réponse"]}</li>";
59.       break;
60.   }
61.   //menu
62.   $modèle->optionsMenu = [
63.     "Liste des simulations" => "main.php?action=lister-simulations",
64.     "Fin de session" => "main.php?action=fin-session"];
65.   // on rend le modèle
66.   return $modèle;
67. }
68. ?>
69. <!-- document HTML -->
70. <!doctype html>
71. <html lang="fr">
72.     <head>
73.         …
74.         <title>Application impots</title>
75.     </head>
76.     <body>
77.         …
78.     </body>
79. </html>
```

Commentaires

- lignes 22-30 : affichage d'un formulaire vide ;
- lignes 31-45 : cas du calcul d'impôt réussi. On réaffiche les valeurs saisies ainsi que le montant de l'impôt ;
- lignes 46-59 : cas de l'échec du calcul d'impôt à cause d'une indisponibilité d'un des serveurs **[Redis]** ou **[MySQL]** ;
- lignes 62-64 : calcul des deux options du menu ;

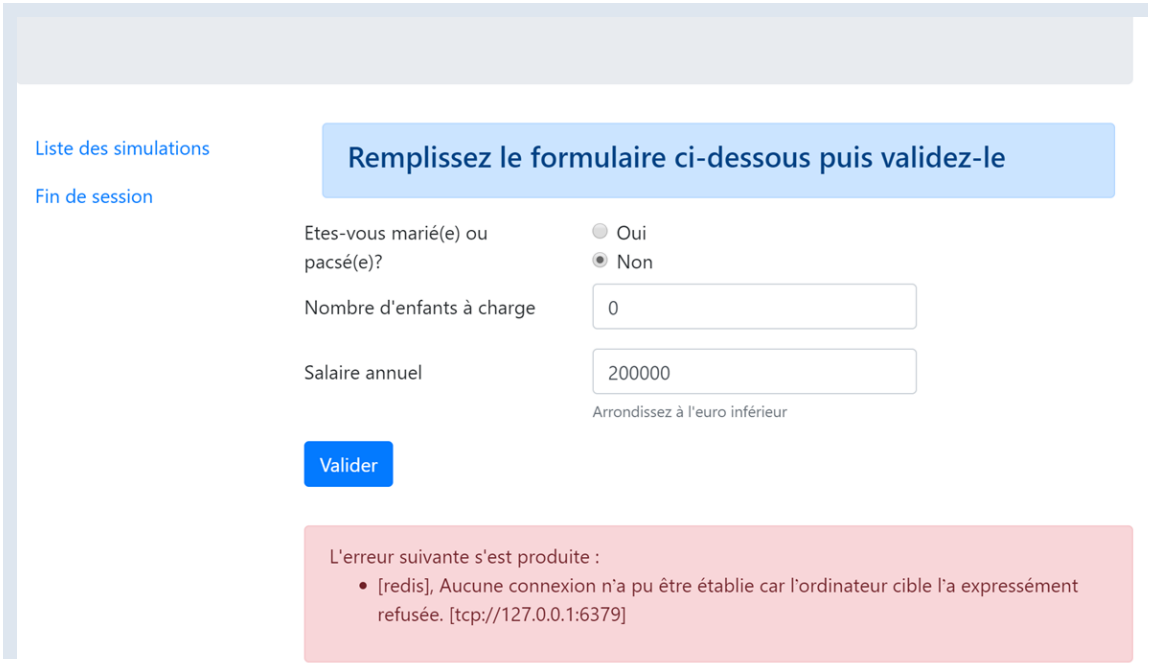
1.23.13.3.6 Tests [Postman]

Le test **[calculer-impot-300]** nous permet d'avoir le code d'état 300. Il correspond à un calcul d'impôt réussi :



- en **[3]**, les valeurs ayant amené au résultat **[2]** ;

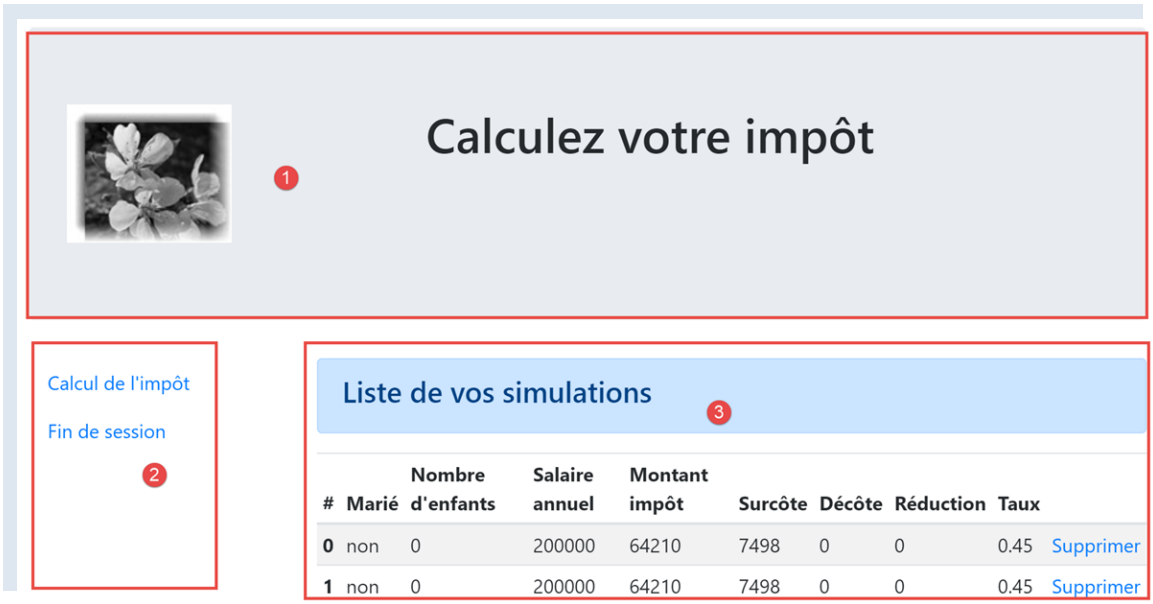
Essayons un cas d'erreur : l'erreur **[350]** dûe à une indisponibilité du serveur **[Redis]** :



1.23.13.4 La vue de la liste des simulations

1.23.13.4.1 Présentation de la vue

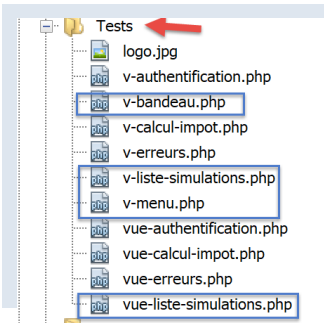
La vue qui présente la liste des simulations est la suivante :



La vue générée par le script **[vue-liste-simulations]** a trois parties :

- 1 : le bandeau supérieur est généré par le fragment **[v-bandeau.php]** déjà présenté ;
- 2 : le tableau des simulations généré par le fragment **[v-liste-simulations.php]** ;
- 3 : un menu présentant deux liens, généré par le fragment **[v-menu.php]** ;

La vue des simulations est générée par le script **[vue-liste-simulations.php]** suivant :




```
1.  <?php
2.
3.  // on calcule le modèle de la vue
4.  $modèle = getModelForThisView();
5.
6.  function getModelForThisView(Request $request, Session $session, array $config, array $content): object {
7.    // on encapsule les données de la pagé dans $modèle
8.    $modèle = new \stdClass();
9.    …
10.   // on rend le modèle
11.   return $modèle;
12. }
13. ?>
14. <!-- document HTML -->
15. <!doctype html>
16. <html lang="fr">
17.     <head>
18.         <!-- Required meta tags -->
19.         <meta charset="utf-8">
20.         <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
21.         <!-- Bootstrap CSS -->
```

```
22.         <link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-
    MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
23.         <title>Application impots</title>
24.     </head>
25.     <body>
26.         <div class="container">
27.             <!-- bandeau -->
28.             <?php require "v-bandeau.php"; ?>
29.             <!-- ligne à deux colonnes -->
30.             <div class="row">
31.                 <!-- menu sur trois colonnes-->
32.                 <div class="col-md-3">
33.                     <?php require "v-menu.php" ?>
34.                 </div>
35.                 <!-- liste des simulations sur 9 colonnes-->
36.                 <div class="col-md-9">
37.                     <?php require "v-liste-simulations.php" ?>
38.                 </div>
39.             </div>
40.         </div>
41.     </body>
42. </html>
```

Commentaires

- ligne 28 : inclusion du bandeau de l'application **[1]** ;
- ligne 33 : inclusion du menu **[2]**. Il sera affiché sur trois colonnes sous le bandeau ;
- ligne 37 : inclusion du tableau des simulations **[3]**. Il sera affiché sur neuf colonnes sous le bandeau et à droite du menu ;

Nous avons déjà commenté deux des trois fragments de cette vue :

- **[v-bandeau.php]** : au paragraphe lien ;
- **[v-menu.php]** : au paragraphe lien ;

Le fragment **[v-liste-simulations.php]** est le suivant :

```
1.  <!-- message sur fond bleu -->
2.  <div class="alert alert-primary" role="alert">
3.      <h4>Liste de vos simulations</h4>
4.  </div>
5.  <!-- tableau des simulations -->
6.  <table class="table table-sm table-hover table-striped">
7.      <!-- entêtes des six colonnes du tableau -->
8.      <thead>
9.          <tr>
10.             <th scope="col">#</th>
11.             <th scope="col">Marié</th>
12.             <th scope="col">Nombre d'enfants</th>
13.             <th scope="col">Salaire annuel</th>
14.             <th scope="col">Montant impôt</th>
15.             <th scope="col">Surcôte</th>
16.             <th scope="col">Décôte</th>
17.             <th scope="col">Réduction</th>
18.             <th scope="col">Taux</th>
19.             <th scope="col"></th>
20.         </tr>
21.     </thead>
22.     <!-- corps du tableau (données affichées) -->
23.     <tbody>
24.         <?php
25.         $i = 0;
26.         // on affiche chaque simulation en parcourant le tableau des simulations
27.         foreach ($modèle->simulations as $simulation) {
28.           // affichage d'une ligne du tableau avec 6 colonnes - balise <tr>
29.           // colonne 1 : entête ligne (n° simulation) - balise <th scope='row'>
30.           // colonne 2 : valeur paramètre [marié] - balise <td>
31.           // colonne 3 : valeur paramètre [enfants] - balise <td>
32.           // colonne 4 : valeur paramètre [salaire] - balise <td>
33.           // colonne 5 : valeur paramètre [impôt] (de l'impôt) - balise <td>
34.           // colonne 6 : valeur paramètre [surcôte] - balise <td>
35.           // colonne 7 : valeur paramètre [décôte] - balise <td>
36.           // colonne 8 : valeur paramètre [réduction] - balise <td>
37.           // colonne 9 : valeur paramètre [taux] (de l'impôt) - balise <td>
```

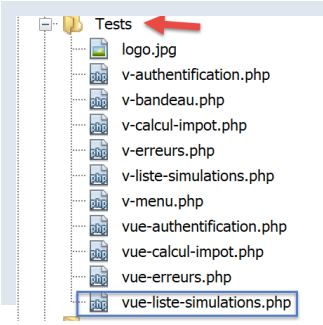
```
38.           // colonne 10 : lien de suppression de la simulation - balise <td>
39.           print <<<EOT
40.         <tr>
41.           <th scope="row">$i</th>
42.           <td>{$simulation["marié"]}</td>
43.           <td>{$simulation["enfants"]}</td>
44.           <td>{$simulation["salaire"]}</td>
45.           <td>{$simulation["impôt"]}</td>
46.           <td>{$simulation["surcôte"]}</td>
47.           <td>{$simulation["décôte"]}</td>
48.           <td>{$simulation["réduction"]}</td>
49.           <td>{$simulation["taux"]}</td>
50.           <td><a href="main.php?action=supprimer-simulation&numéro=$i">Supprimer</a></td>
51.         </tr>
52. EOT;
53.           $i++;
54.         }
55.         ?>
56.         </tr>
57.     </tbody>
58. </table>
```

Commentaires

- un tableau HTML est réalisé avec la balise <table> (lignes 6 et 58) ;
- les entêtes des colonnes du tableau se font à l'intérieur d'une balise <thead> (table head, lignes 8, 21). La balise <tr> (*table row*, lignes 9 et 20) délimitent une ligne. Lignes 10-15, la balise <th> (*table header*) définit un entête de colonne. Il y en a donc dix. **[scope="col"]** indique que l'entête s'applique à la colonne. **[scope="row"]** indique que l'entête s'applique à la ligne ;
- lignes 23-57 : la balise <tbody> encadre les données affichées par le tableau ;
- lignes 40-51 : la balise <tr> encadre une ligne du tableau ;
- ligne 41 : la balise <th scope='row'> définit l'entête de la ligne ;
- lignes 42-50 : chaque balise <td> définit une colonne de la ligne ;
- ligne 27 : la liste des simulations sera trouvée dans le modèle **[$modèle→simulations]** qui est un tableau associatif ;
- ligne 50 : un lien pour supprimer la simulation. L'URL reprend le n° affiché dans la 1re colonne du tableau (ligne 41) ;

1.23.13.4.2 Test visuel

Nous rassemblons ces différents éléments dans le dossier **[Tests]** et nous créons un modèle de test pour la vue **[vue-liste-simulations.php]** :



Le modèle de données de la vue **[vue-liste-simulations]** sera le suivant :

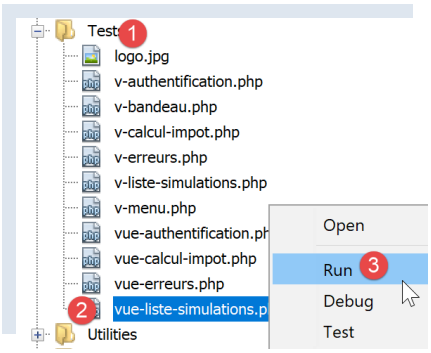
```
1.  <?php
2.  // on calcule le modèle de la vue
3.  $modèle = getModelForThisView();
4.
5.  function getModelForThisView(): object {
6.    // on encapsule les données de la pagé dans $modèle
7.    $modèle = new \stdClass();
8.    // on met les simulations au format attendu par la page
9.    $modèle->simulations = [
10.     [
11.       "marié" => "oui",
12.       "enfants" => 2,
13.       "salaire" => 60000,
14.       "impôt" => 448,
15.       "décôte" => 100,
16.       "réduction" => 20,
17.       "surcôte" => 0,
```

```
18.       "taux" => 0.14
19.     ],
20.     [
21.       "marié" => "non",
22.       "enfants" => 2,
23.       "salaire" => 200000,
24.       "impôt" => 25600,
25.       "décôte" => 0,
26.       "réduction" => 0,
27.       "surcôte" => 8400,
28.       "taux" => 0.45
29.     ]
30.   ];
31.   // les options de menu
32.   $modèle->optionsMenu = [
33.     "Calcul de l'impôt" => "main.php?action=afficher-calcul-impot",
34.     "Fin de session" => "main.php?action=fin-session"];
35.   // image du bandeau
36.   $modèle->logo = "http://localhost/php7/scripts-web/impots/version-12/Tests/logo.jpg";
37.   // on rend le modèle
38.   return $modèle;
39. }
40. ?>
41. <!-- document HTML -->
42. <!doctype html>
43. <html lang="fr">
44.     <head>
45.         …
46.     </head>
47.     <body>
48.         …
49.     </body>
50. </html>
```

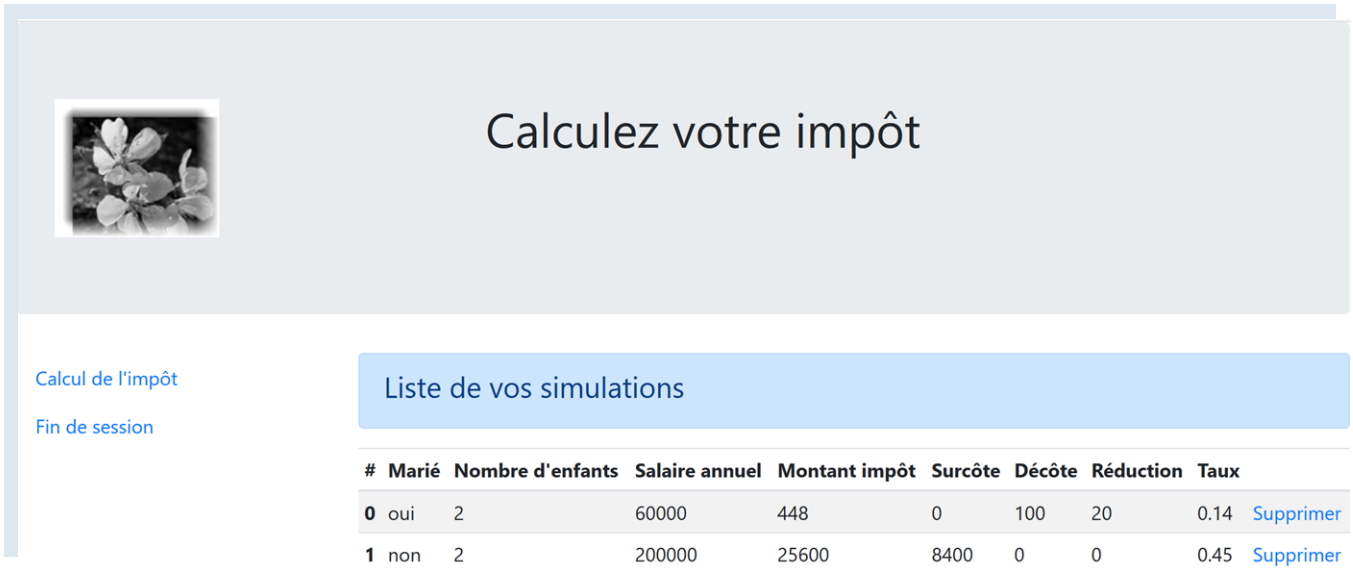
Commentaires

- lignes 9-30 : le tableau des simulations affichées par la table HTML ;
- lignes 32-34 : le tableau des options de menu ;

Affichons cette vue :

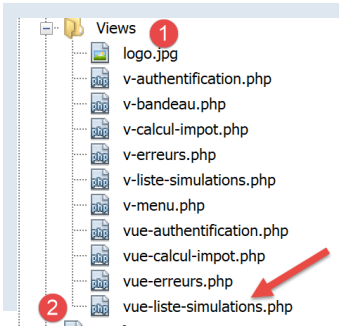


On obtient le résultat suivant :



On travaille sur cette vue jusqu'à ce que le résultat obtenu visuellement nous convienne. On peut ensuite passer à l'intégration de la vue dans l'application web en cours d'écriture.

1.23.13.4.3 Calcul du modèle de la vue



Une fois l'aspect visuel de la vue déterminé, on peut procéder au calcul du modèle de la vue en conditions réelles. Rappelons les codes d'état qui mènent à cette vue. On les trouve dans le fichier de configuration :

```
1.  "vues": {
2.          "vue-authentification.php": [700, 221, 400],
3.          "vue-calcul-impot.php": [200, 300, 341, 350, 800],
4.          "vue-liste-simulations.php": [500, 600]
5.      },
6.  "vue-erreurs": "vue-erreurs.php"
```

Ce sont donc les codes d'état **[500, 600]** qui font afficher la vue des simulations. Pour retrouver la signification de ces codes, on peut s'aider des tests **[Postman]** réalisés sur l'application jSON :

- **[lister-simulations-500]** : 500 est le code d'état à l'issue d'une action **[lister-simulations]** réussie : on présente alors la liste des simulations réalisées par l'utilisateur ;
- **[supprimer-simulation-600]** : 600 est le code d'état à l'issue d'une action **[supprimer-simulation]** réussie. On présente alors la nouvelle liste des simulations obtenue après cette suppression ;

Maintenant que nous savons à quels moments doit être affichée la liste des simulations, on peut calculer son modèle dans **[vue-liste-simulations.php]** :

```
1.  <?php
2.  // on hérite des variables suivantes
3.  // Request $request : la requête en cours
4.  // Session $session : la session de l'application
5.  // array $config : la configuration de l'application
6.  // array $content : la réponse du contrôleur
7.  // pas d'erreurs possibles
```

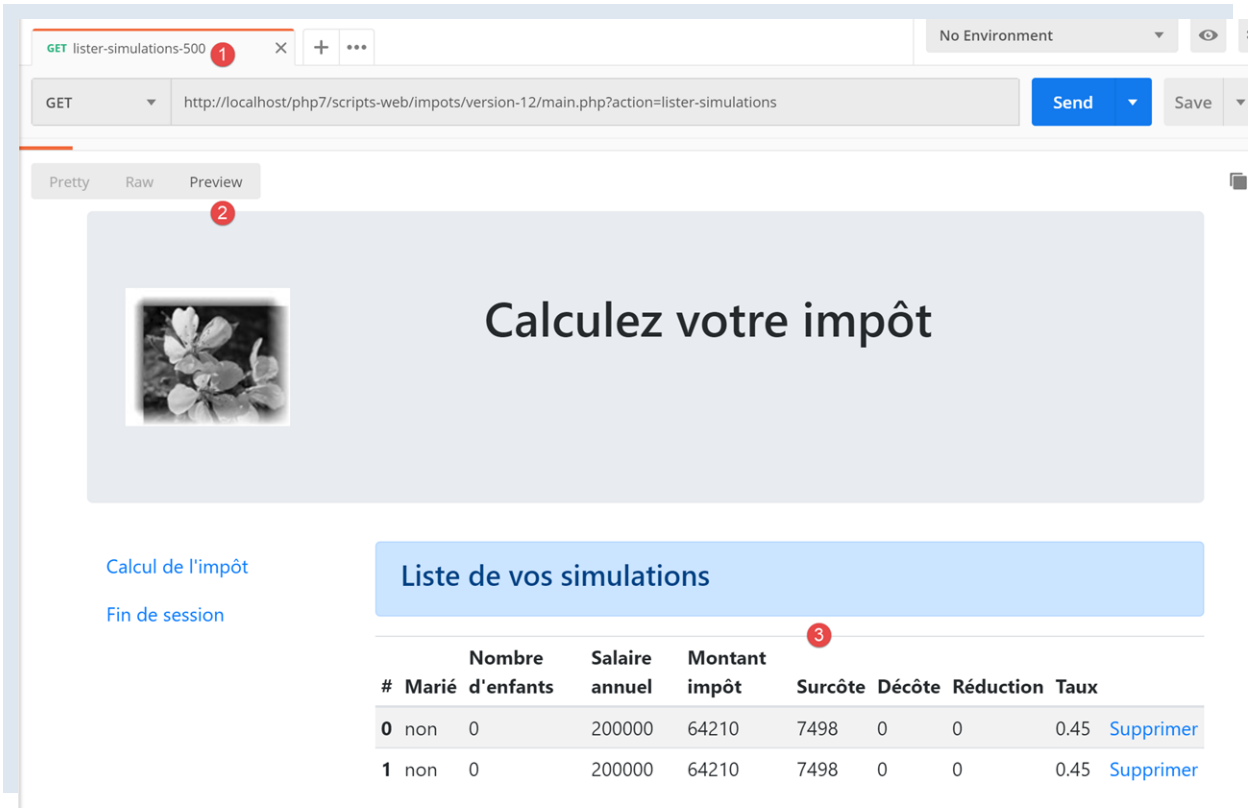
```
8.  // array $content : la réponse du contrôleur
9.  //
10. // dépendances Symfony
11. use Symfony\Component\HttpFoundation\Request;
12. use Symfony\Component\HttpFoundation\Session\Session;
13.
14. // on calcule le modèle de la vue
15. $modèle = getModelForThisView($request, $session, $config, $content);
16.
17. function getModelForThisView(Request $request, Session $session, array $config, array $content): object {
18.   // on encapsule les données de la pagé dans $modèle
19.   $modèle = new \stdClass();
20.   // on met les simulations au format attendu par la page
21.   // elles sont trouvées dans la réponse du contrôleur qui a exécuté l'action
22.   // sous la forme d'un tableau d'objets de type [Simulation]
23.   $objetsSimulation = $content["réponse"];
24.   // chaque objet [Simulation] va être transformé en tableau associatif
25.   $modèle->simulations = [];
26.   foreach ($objetsSimulation as $objetSimulation) {
27.     $modèle->simulations[] = [
28.       "marié" => $objetSimulation->getMarié(),
29.       "enfants" => $objetSimulation->getEnfants(),
30.       "salaire" => $objetSimulation->getSalaire(),
31.       "impôt" => $objetSimulation->getImpôt(),
32.       "surcôte" => $objetSimulation->getSurcôte(),
33.       "décôte" => $objetSimulation->getdécôte(),
34.       "réduction" => $objetSimulation->getRéduction(),
35.       "taux" => $objetSimulation->getTaux()
36.     ];
37.   }
38.   // les options de menu
39.   $modèle->optionsMenu = [
40.     "Calcul de l'impôt" => "main.php?action=afficher-calcul-impot",
41.     "Fin de session" => "main.php?action=fin-session"];
42.   // on rend le modèle
43.   return $modèle;
44. }
45. ?>
46. <!-- document HTML -->
47. <!doctype html>
48. <html lang="fr">
49.     <head>
50.         …
51.     </head>
52.     <body>
53.         …
54.     </body>
55. </html>
```

Commentaires

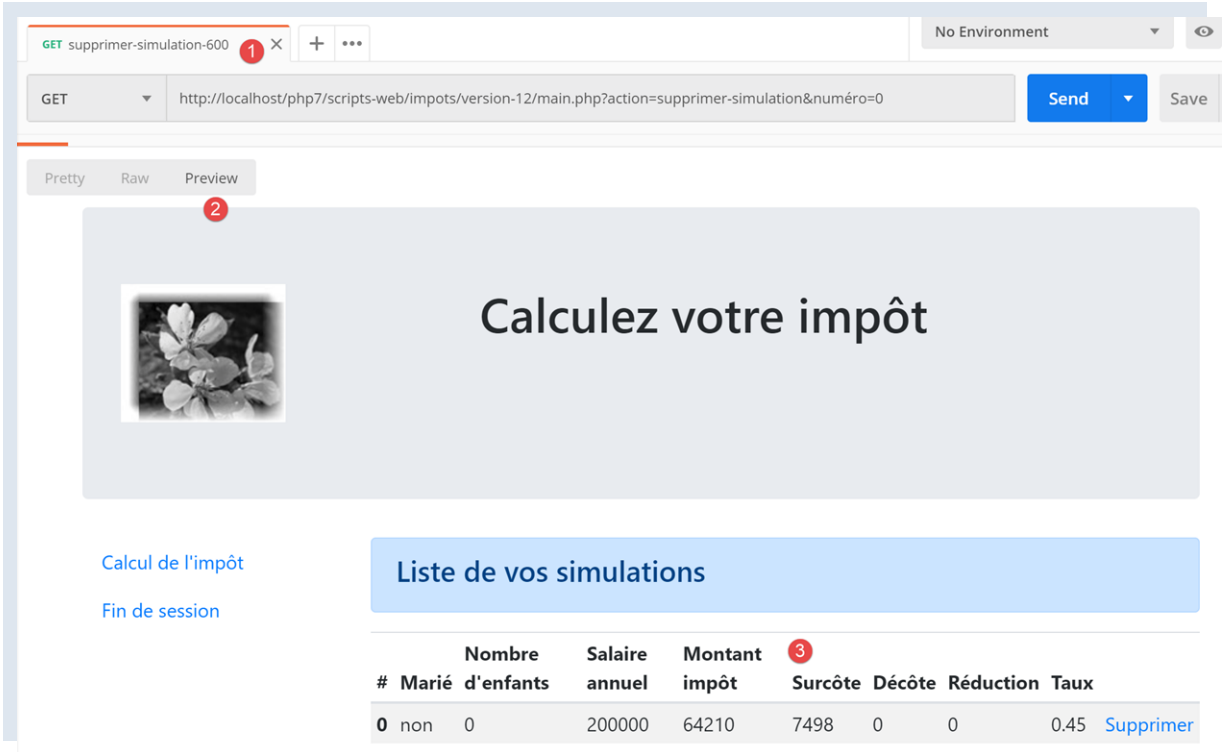
- lignes 26-36 : calcul du modèle **[$modèle→simulations]** utilisé par le fragment **[v-liste-simulations.php]** ;
- lignes 39-41 : calcul du modèle **[$modèle→optionsMenu]** utilisé par le fragment **[v-menu.php]** ;

1.23.13.4.4 Tests [Postman]

Le test **[lister-simulations-500]** nous permet d'avoir le code d'état 500. Il correspond à une demande pour voir les simulations :



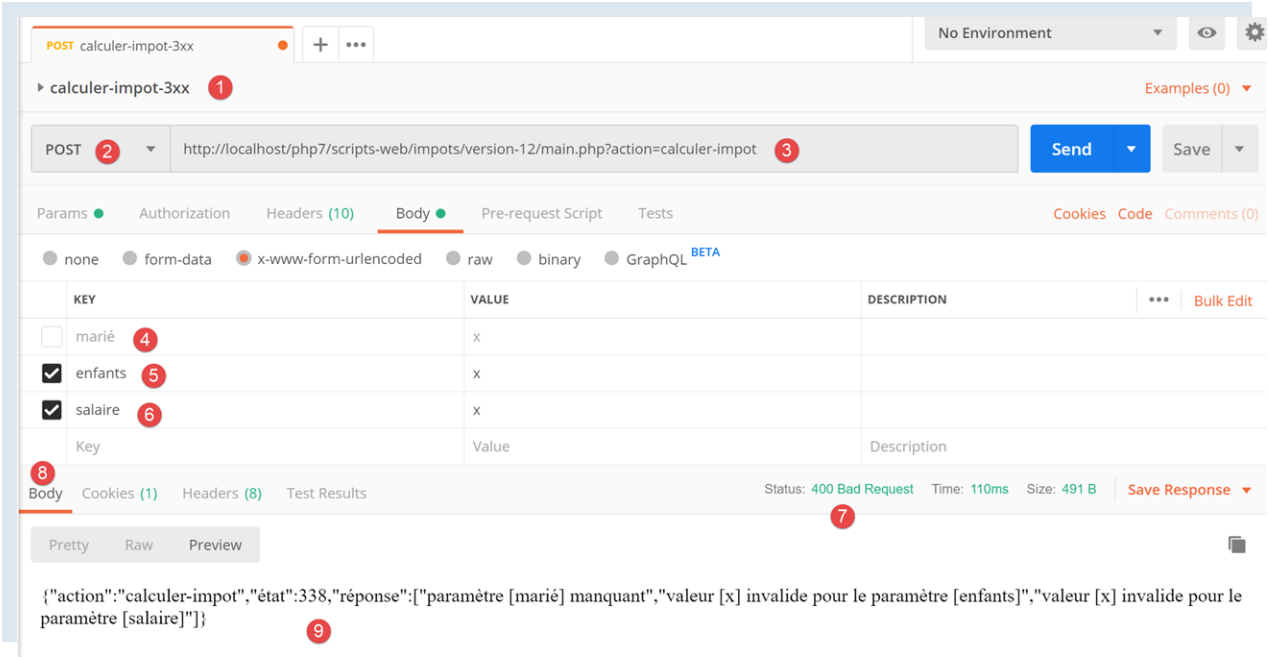
Le test **`[supprimer-simulation-600]`** nous permet d'avoir le code d'état 600. Il correspond à la suppression réussie de la simulation n° 0. Le résultat renvoyé est une liste de simulations avec une simulation en moins :



1.23.13.5 La vue des erreurs inattendues

On appelle ici, erreur inattendue, une erreur qui n'aurait pas dû se produire dans le cadre d'une utilisation normale de l'application web.

Prenons comme exemple, le test **[Postman] [calculer-impot-3xx]** défini comme suit :



- en **[1-3]**, une requête POST avec l'action **[calculer-impot]** ;
- en **[4-6]** : ici on peut définir ce qu'on veut pour les trois paramètres du POST :
  - **[4]** : le paramètre **[marié]** est manquant ;
  - **[5-6]** : les paramètres **[enfants, salaire]** sont présents mais invalides ;
- en **[9]**, ces trois erreurs sont signalées avec le code d'état 338 ;

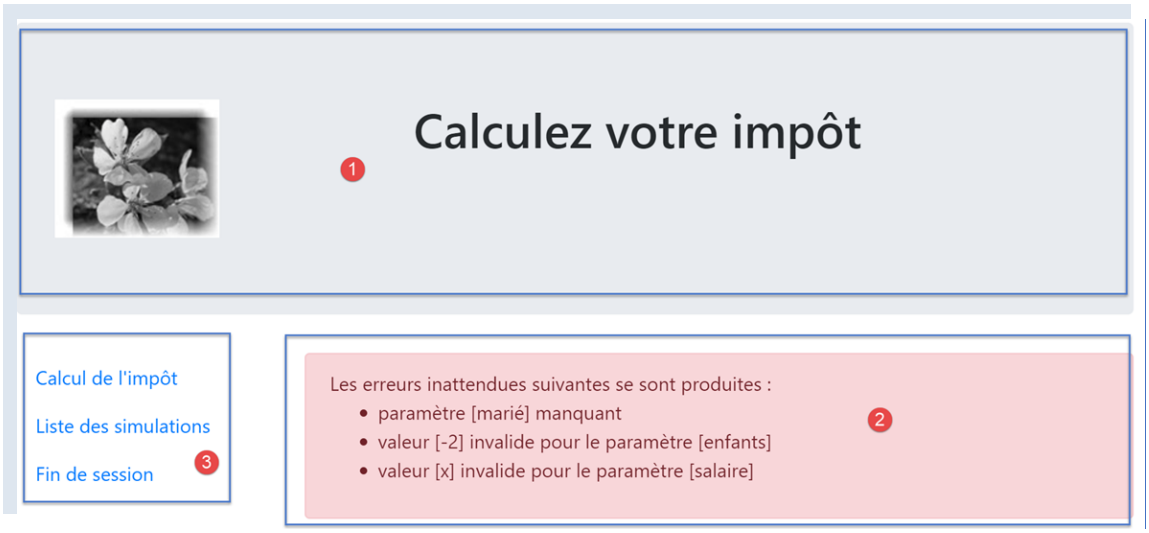
Or dans le formulaire HTML de l'application web, ce cas ne peut pas se produire :

- tous les paramètres sont présents ;
- le paramètre **[marié]** qui prend sa valeur dans les attributs **[value]** de deux boutons radio, a forcément l'une des valeurs **[oui]** ou **[non]** ;
- avec un navigateur récent, les attributs <input type='number' min='0' step='1' …> font que les saisies pour les enfants et le salaire sont forcément des nombres entiers >=0 ;

Cependant, rien n'empêche un utilisateur de prendre **[Postman]** et d'envoyer à notre serveur, le test **[calcul-impot-3xx]** ci-dessus. On a vu que notre application web savait répondre correctement à cette requête. On appellera, **erreur inattendue**, une erreur qui ne devrait pas se produire dans le cadre de l'application HTML. Si elle se produit, c'est que probablement quelqu'un essaie de 'hacker' l'application. Par souci de pédagogie, on a décidé d'afficher une vue d'erreurs pour ces cas. Dans la réalité, on pourrait réafficher la dernière page envoyée au client. Il suffit pour cela d'enregistrer en session, la dernière réponse HTML envoyée. En cas d'erreur inattendue, on renvoie cette réponse. Ainsi l'utilisateur aura l'impression que le serveur ne répond pas à ses erreurs puisque la page affichée ne change pas.

1.23.13.5.1 Présentation de la vue

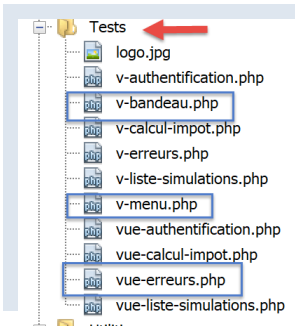
La vue qui présente les erreurs inattendues est la suivante :



La vue générée par le script **[vue-erreurs.php]** a trois parties :

- 1 : le bandeau supérieur est généré par le fragment **[v-bandeau.php]** déjà présenté ;
- 2 : la ou les erreurs inattendues ;
- 3 : un menu présentant trois liens, généré par le fragment **[v-menu.php]** ;

La vue des erreurs inattendues est générée par le script **[vue-erreurs.php]** suivant :




```
1.  <?php
2.  // on calcule le modèle de la vue
3.  $modèle = getModelForThisView();
4.
5.  function getModelForThisView(): object {
6.    // on encapsule les données de la pagé dans $modèle
7.    $modèle = new \stdClass();
8.  …
9.    // on retourne le modèle
10.   return $modèle;
11. }
12. ?>
13. <!-- document HTML -->
14. <!doctype html>
15. <html lang="fr">
16.     <head>
17.         <!-- Required meta tags -->
18.         <meta charset="utf-8">
19.         <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
20.         <!-- Bootstrap CSS -->
21.         <link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-
    MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
22.         <title>Application impots</title>
23.     </head>
24.     <body>
```

```
25.         <div class="container">
26.             <!-- bandeau sur 12 colonnes -->
27.             <?php require "v-bandeau.php"; ?>
28.             <!-- ligne à deux colonnes -->
29.             <div class="row">
30.                 <!-- menu sur 3 colonnes-->
31.                 <div class="col-md-3">
32.                     <?php require "v-menu.php" ?>
33.                 </div>
34.                 <!-- liste des erreurs -->
35.                 <div class="col-md-9">
36.                     <?php
37.                     print <<<EOT
38.                       <div class="alert alert-danger" role="alert">
39.                         Les erreurs inattendues suivantes se sont produites :
40.                         <ul>$modèle->erreurs</ul>
41.                       </div>
42. EOT;
43.                     ?>
44.                 </div>
45.             </div>
46.         </div>
47.     </body>
48. </html>
```

Commentaires

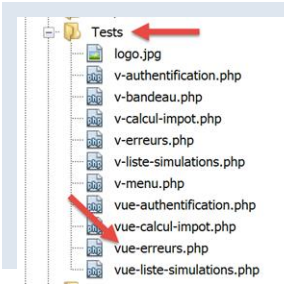
- ligne 27 : inclusion du bandeau de l'application **[1]** ;
- ligne 32 : inclusion du menu **[2]**. Il sera affiché sur trois colonnes sous le bandeau ;
- lignes 34-44 : affichage de la zone d'erreurs sur neuf colonnes ;
- lignes 37-44 : l'opération **[print]** qui affiche les erreurs inattendues ;
- ligne 38 : cet affichage se fera dans un cadre Bootstrap à fond rose ;
- ligne 39 : un texte de présentation ;
- ligne 40 : la balise <ul> encadre une liste à puces. Cette liste à puces est fournie par le modèle **[$modèle->erreurs]** ;

Nous avons déjà commenté les deux fragments de cette vue :

- **[v-bandeau.php]** : au paragraphe lien ;
- **[v-menu.php]** : au paragraphe lien ;

1.23.13.5.2 Test visuel

Nous rassemblons ces différents éléments dans le dossier **[Tests]** et nous créons un modèle de test pour la vue **[vue-erreurs.php]** :



Le modèle de données de la vue **[vue-erreurs.php]** sera le suivant :

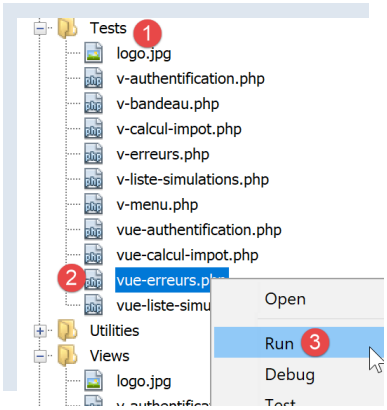
```
1.  <?php
2.  // on calcule le modèle de la vue
3.  $modèle = getModelForThisView();
4.
5.  function getModelForThisView(): object {
6.    // on encapsule les données de la pagé dans $modèle
7.    $modèle = new \stdClass();
8.
9.    // le tableau des erreurs inattendues
10.   $erreurs = ["erreur1", "erreur2"];
11.   // on construit la liste HTML des erreurs
12.   $modèle->erreurs = "";
13.   foreach ($erreurs as $erreur) {
14.     $modèle->erreurs .= "<li>$erreur</li>";
```

```
15.    }
16.    // options du menu
17.    $modèle->optionsMenu = [
18.      "Calcul de l'impôt" => "main.php?action=afficher-calcul-impot",
19.      "Liste des simulations" => "main.php?action=lister-simulations",
20.      "Fin de session" => "main.php?action=fin-session",];
21.    // image du bandeau
22.    $modèle->logo = "http://localhost/php7/scripts-web/impots/version-12/Tests/logo.jpg";
23.    // on retourne le modèle
24.    return $modèle;
25. }
26. ?>
27. <!-- document HTML -->
28. <!doctype html>
29. <html lang="fr">
30.     <head>
31.         …
32.     </head>
33.     <body>
34.         …
35.     </body>
36. </html>
```

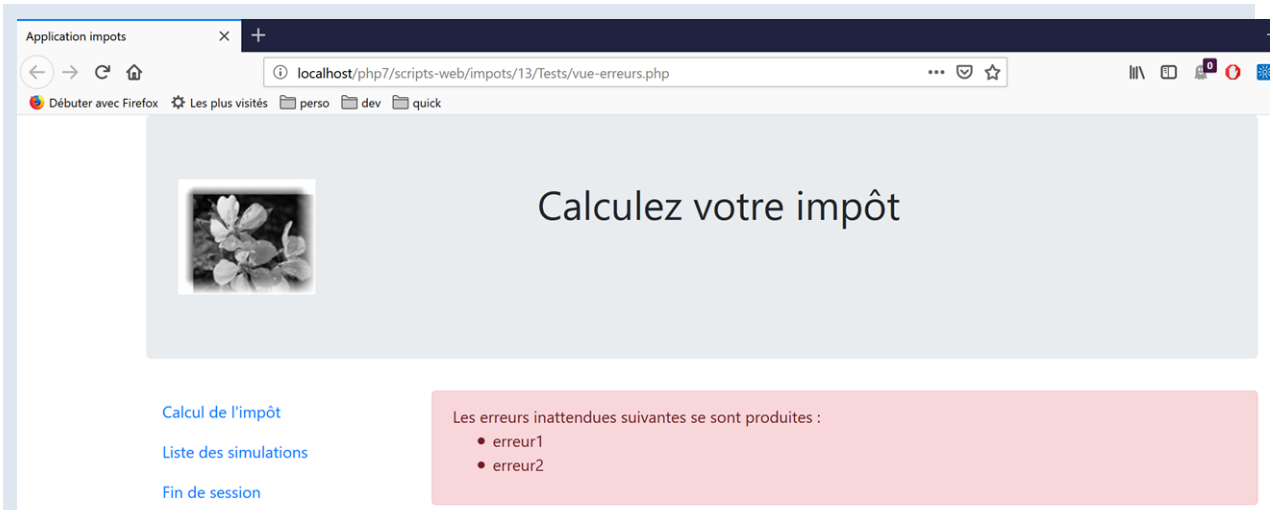
Commentaires

- lignes 9-15 : construction de la liste HTML des erreurs ;
- lignes 17-20 : le tableau des options de menu ;

Affichons cette vue :

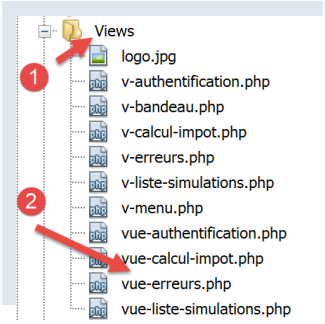


On obtient le résultat suivant :



On travaille sur cette vue jusqu'à ce que le résultat obtenu visuellement nous convienne. On peut ensuite passer à l'intégration de la vue dans l'application web en cours d'écriture.

1.23.13.5.3 Calcul du modèle de la vue



Une fois l'aspect visuel de la vue déterminé, on peut procéder au calcul du modèle de la vue en conditions réelles. Rappelons les codes d'état qui mènent à cette vue. On les trouve dans le fichier de configuration :

```
1.  "vues": {
2.          "vue-authentification.php": [700, 221, 400],
3.          "vue-calcul-impot.php": [200, 300, 341, 350, 800],
4.          "vue-liste-simulations.php": [500, 600]
5.      },
6.  "vue-erreurs": "vue-erreurs.php"
```

Ce sont donc les codes d'état qui ne sont pas dans ceux des lignes **[2-4]** qui font afficher la vue des erreurs inattendues.

Le code de calcul du modèle de la vue **[vue-erreurs.php]** est le suivant :

```
1.  <?php
2.  // on hérite des variables suivantes
3.  // Request $request : la requête en cours
4.  // Session $session : la session de l'application
5.  // array $config : la configuration de l'application
6.  // array $content : la réponse du contrôleur
7.  //
8.  // dépendances Symfony
9.  use Symfony\Component\HttpFoundation\Request;
10. use Symfony\Component\HttpFoundation\Session\Session;
11.
12. // on calcule le modèle de la vue
13. $modèle = getModelForThisView($request, $session, $config, $content);
14.
15. function getModelForThisView(Request $request, Session $session, array $config, array $content): object {
16.   // on encapsule les données de la pagé dans $modèle
17.   $modèle = new \stdClass();
18.
19.   // on récupère les erreurs dans la réponse du contrôleur
20.   $réponse = $content["réponse"];
21.   if (!is_array($réponse)) {
22.     // un seul message d'erreur
23.     $erreurs = [$réponse];
24.   } else {
25.     // plusieurs messages d'erreur
26.     $erreurs = $réponse;
27.   }
28.   // on construit la liste HTML des erreurs
29.   $modèle->erreurs = "";
30.   foreach ($erreurs as $erreur) {
31.     $modèle->erreurs .= "<li>$erreur</li>";
32.   }
33.   // options du menu
34.   $modèle->optionsMenu = [
35.     "Calcul de l'impôt" => "main.php?action=afficher-calcul-impot",
36.     "Liste des simulations" => "main.php?action=lister-simulations",
37.     "Fin de session" => "main.php?action=fin-session",];
38.
39.   // on retourne le modèle
40.   return $modèle;
41. }
42. ?>
```

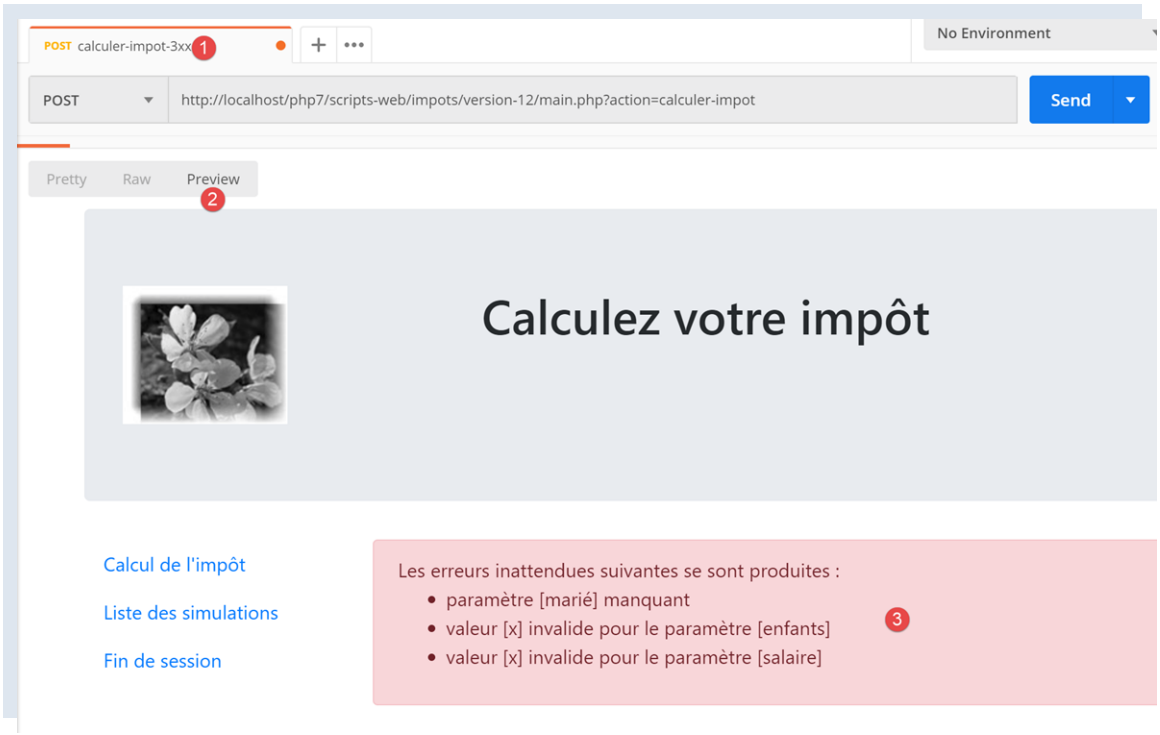
```
43. <!-- document HTML -->
44. <!doctype html>
45. <html lang="fr">
46.     <head>
47.         …
48.     </head>
49.     <body>
50.         …
51.     </body>
52. </html>
```

Commentaires

- lignes 19-32 : calcul du modèle **[$modèle→erreurs]** utilisé par la vue **[vue-erreurs.php]** ;
- lignes 34-37 : calcul du modèle **[$modèle→optionsMenu]** utilisé par le fragment **[v-menu.php]** ;

1.23.13.5.4 Tests [Postman]

Le test **[calculer-impot-3xx]** nous permet d'avoir le code d'état 338 qui n'est pas un code d'état attendu. La réponse HTML est alors la suivante :



1.23.13.6 Implémentation des actions du menu de l'application

Nous allons ici traiter de l'implémentation des actions du menu. Rappelons la signification des liens que nous avons rencontrés

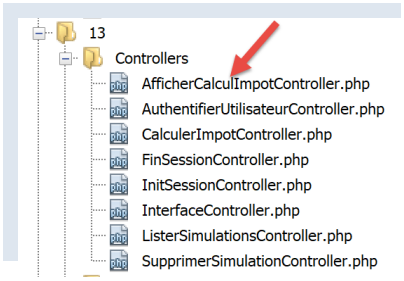
Vue	Lien	Cible	Rôle
Calcul de l'impôt	**[Liste des simulations]**	**[main.php?action=lister-simulations]**	Demander la liste des simulations
	[Fin de session]	**[main.php?action=fin-session]**	Demander la fin de la session
Liste des simulations	**[Calcul de l'impôt]**	**[main.php?action=afficher-calcul-impot]**	Afficher la vue du calcul d'impôt
	[Fin de session]	**[main.php?action=fin-session]**	Demander la fin de la session
	[Calcul de l'impôt]	**[main.php?action=afficher-calcul-impot]**	Afficher la vue du calcul d'impôt
Erreurs inattendues	**[Liste des simulations]**	**[main.php?action=lister-simulations]**	Demander la liste des simulations
	[Fin de session]	**[main.php?action=fin-session]**	Demander la fin de la session

Il faut rappeler qu'un clic sur un lien provoque un GET vers la cible du lien. Les actions **[lister-simulations, fin-session]** ont été implémentées avec une opération GET, ce qui nous permet de les mettre comme cibles de liens. Lorsque l'action se fait par un POST, l'utilisation d'un lien n'est plus possible sauf à l'associer avec du Javascript.

Des actions ci-dessus, il apparaît que l'action **[afficher-calcul-impot]** n'a pas encore été implémentée. C'est une opération de navigation entre deux vues : le serveur jSON ou XML n'a aucune raison de l'implémenter car ils n'ont pas la notion de vue. C'est le serveur HTML qui introduit cette notion.

Il nous faut donc implémenter l'action **[afficher-calcul-impot]**. Cela va nous permettre de réviser le mode opératoire de l'implémentation d'une action au sein du serveur.

Tout d'abord, il nous faut ajouter un nouveau contrôleur secondaire. Nous l'appellerons **[AfficherCalculImpotController]** :



Ce contrôleur doit être ajouté dans le fichier de configuration **[config.json]** :

```
1.  {
2.      "databaseFilename": "database.json",
3.      "rootDirectory": "C:/myprograms/Laragon-Lite/www/php7/scripts-web/impots/version-12",
4.      "relativeDependencies": [
5.
6.          …
7.
8.          "/Controllers/InterfaceController.php",
9.          "/Controllers/InitSessionController.php",
10.         "/Controllers/ListerSimulationsController.php",
11.         "/Controllers/AuthentifierUtilisateurController.php",
12.         "/Controllers/CalculerImpotController.php",
13.         "/Controllers/SupprimerSimulationController.php",
14.         "/Controllers/FinSessionController.php",
15.         "/Controllers/AfficherCalculImpotController.php"
16.     ],
17.     "absoluteDependencies": [
18.         "C:/myprograms/Laragon-Lite/www/vendor/autoload.php",
19.         "C:/myprograms/Laragon-Lite/www/vendor/predis/predis/autoload.php"
20.     ],
21.     …
22.     "actions":
23.             {
24.                 "init-session": "\\InitSessionController",
25.                 "authentifier-utilisateur": "\\AuthentifierUtilisateurController",
26.                 "calculer-impot": "\\CalculerImpotController",
27.                 "lister-simulations": "\\ListerSimulationsController",
28.                 "supprimer-simulation": "\\SupprimerSimulationController",
29.                 "fin-session": "\\FinSessionController",
30.                 "afficher-calcul-impot": "\\AfficherCalculImpotController"
31.             },
32.     …
33.     "vues": {
34.         "vue-authentification.php": [700, 221, 400],
35.         "vue-calcul-impot.php": [200, 300, 341, 350, 800],
36.         "vue-liste-simulations.php": [500, 600]
37.     },
38.     "vue-erreurs": "vue-erreurs.php"
39. }
```

- ligne 15 : le nouveau contrôleur ;
- ligne 30 : la nouvelle action et son contrôleur ;
- ligne 35 : le nouveau contrôleur rendra le code d'état 800. Sur un changement de vues, il ne peut y avoir d'erreur ;

Le contrôleur **[AfficherCalculImpotController.php]** sera le suivant :

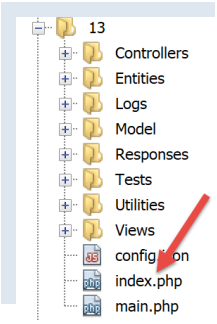
```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use Symfony\Component\HttpFoundation\Request;
7.  use Symfony\Component\HttpFoundation\Session\Session;
8.  use Symfony\Component\HttpFoundation\Response;
9.
10. class AfficherCalculImpotController implements InterfaceController {
11.
12.   // $config est la configuration de l'application
13.   // traitement d'une requête Request
14.   // utile la session Session et peut la modifier
15.   // $infos sont des informations supplémentaires propres à chaque contrôleur
16.   // rend un tableau [$statusCode, $état, $content, $headers]
17.
18.   public function execute(
19.     array $config,
20.     Request $request,
21.     Session $session,
22.     array $infos = NULL): array {
23.
24.     // changement de vue - juste un code d'état à positionner
25.     return [Response::HTTP_OK, 800, ["réponse" => ""], []];
26.   }
27.
28. }
```

Commentaires

- ligne 10 : comme les autres contrôleurs secondaires, le nouveau contrôleur implémente l'interface **[InterfaceController]** ;
- les changements de vue sont simples à implémenter : il suffit de rendre le code d'état associé à la vue cible, ici le code 800 comme il a été vu plus haut ;

1.23.13.7 Tests en conditions réelles

Le code a été écrit et chaque action testée avec **[Postman]**. Il nous reste à tester l'enchaînement des vues en situation réelle. Il nous faut un moyen d'initialiser la session HTML. On sait qu'il faut envoyer au serveur les paramètres **[action=init-session&type=html]**. Pour éviter d'avoir à les taper dans la zone d'adresse du navigateur, on va ajouter le script **[index.php]** à notre application :



Le script **[index.php]** sera le suivant :

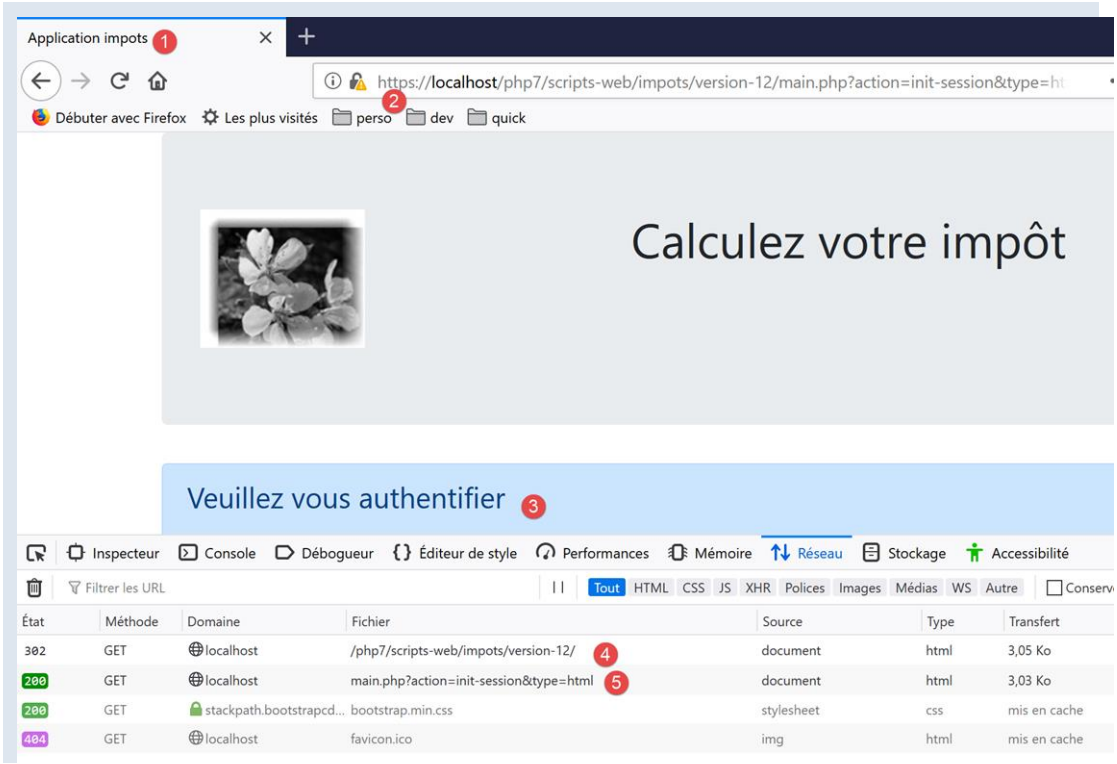
```
1.  <?php
2.
3.  // redirection vers [main.php] en mode [html]
4.  header('Location: main.php?action=init-session&type=html');
```

- ligne 4 : **[header]** est une fonction PHP ajoutant un entête HTTP à la réponse. L'entête HTTP **[Location: main.php?action=init-session&type=html]** demande au navigateur client de se rediriger vers l'URL cible indiquée dans **[Location]**. Le script **[index.php]** est demandé avec l'URL **[http://localhost/php7/scripts-web/impots/version-12/index.php]**. Lorsque le navigateur client va recevoir la redirection vers l'URL relative **[main.php?action=init-session&type=html]**, il va demander l'URL absolue **[http://localhost/php7/scripts-web/impots/version-12/main.php?action=init-session&type=html]** et la session HTML va démarrer ;

L'URL de démarrage peut être simplifiée en **[http://localhost/php7/scripts-web/impots/version-12/]**. Dans le cas où aucune page n'est précisée dans l'URL, les pages **[index.html, index.php]** sont utilisées par défaut. Ici le script **[index.php]** sera donc utilisé ;

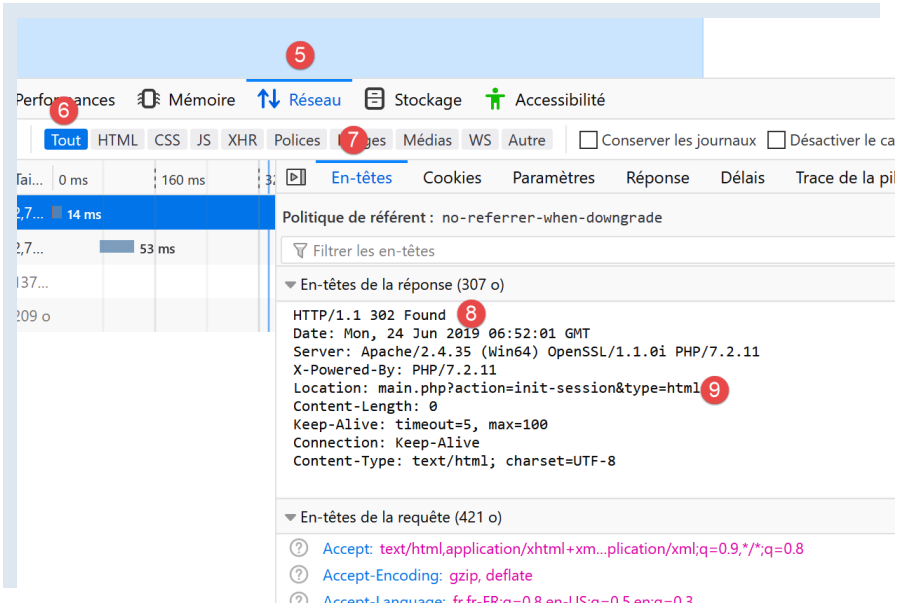
Allons y : nous présentons maintenant quelques enchaînements de vues.

Dans notre navigateur, nous activons le suivi des requêtes (F12 sur Firefox) et nous demandons l'URL de démarrage **[https://localhost/php7/scripts-web/impots/version-12/]** :

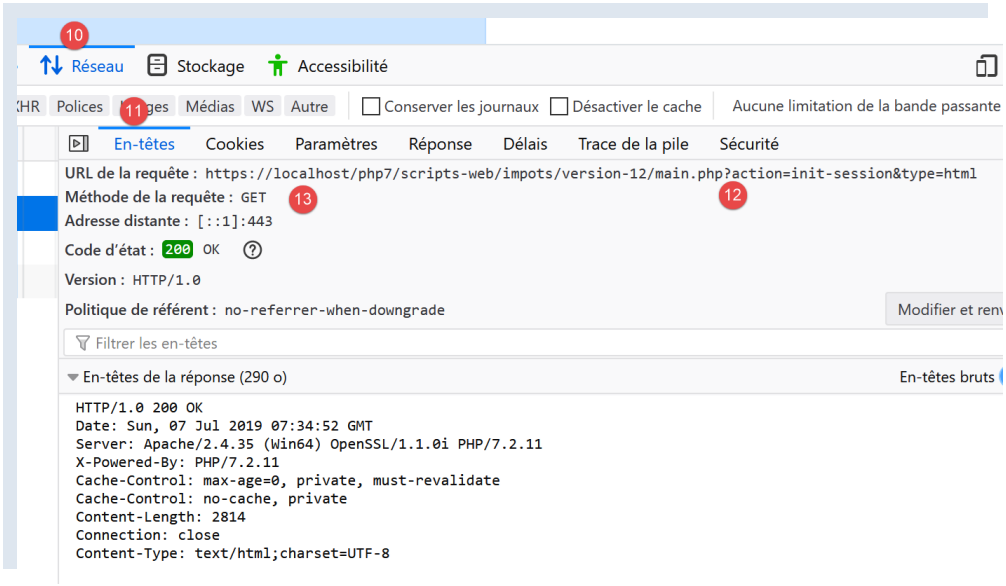


- en **[4]**, la 1re réponse du serveur est une redirection 302 :
- en **[5]**, une nouvelle requête est faite vers l'URL **[http://localhost/php7/scripts-web/impots/13/main.php?action=init-session&type=html]** ;

Regardons de plus près, la redirection 302 :

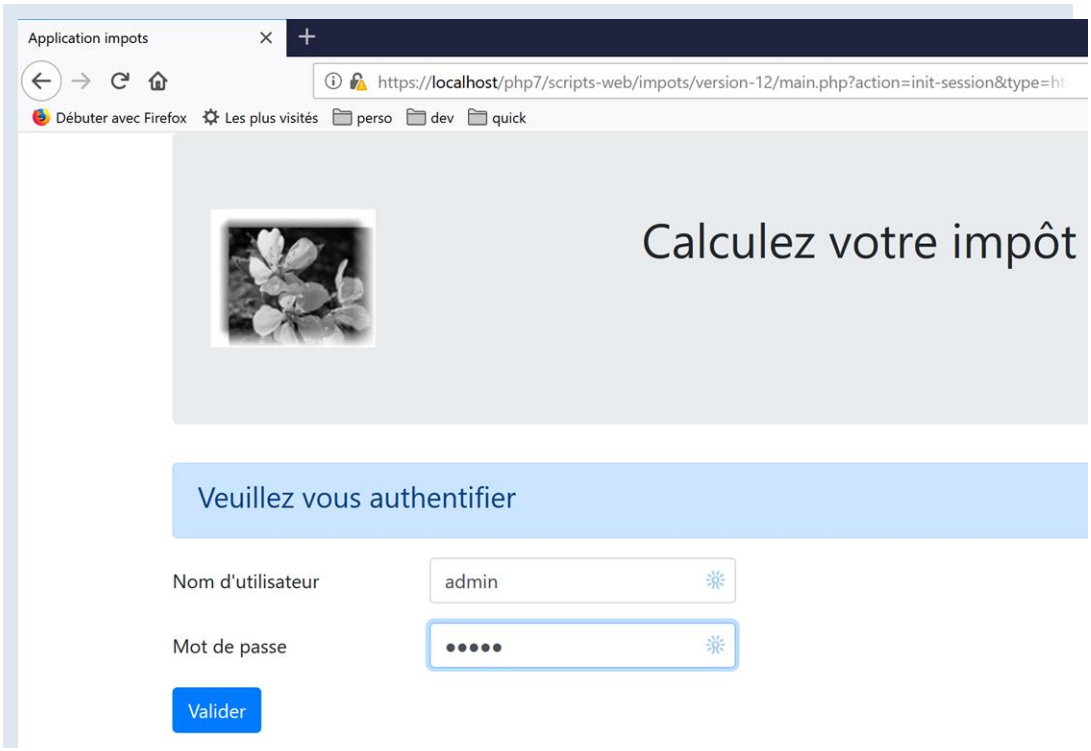


- en **[8]**, le code HTTP **[302]** est un code de redirection : on dit au navigateur client que l'URL demandée a été déplacée. La nouvelle URL est précisée en **[9]**. Le navigateur va suivre cette redirection avec une nouvelle requête GET :

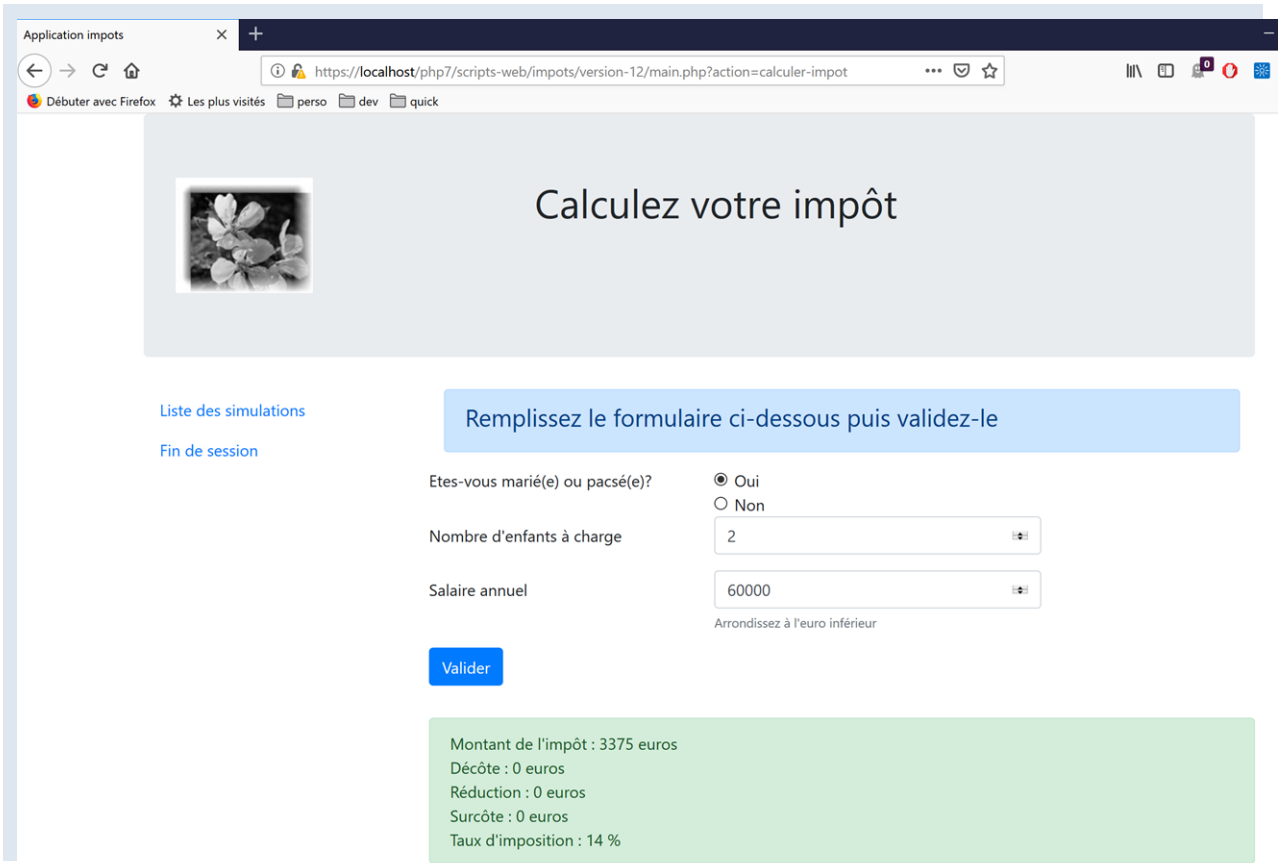


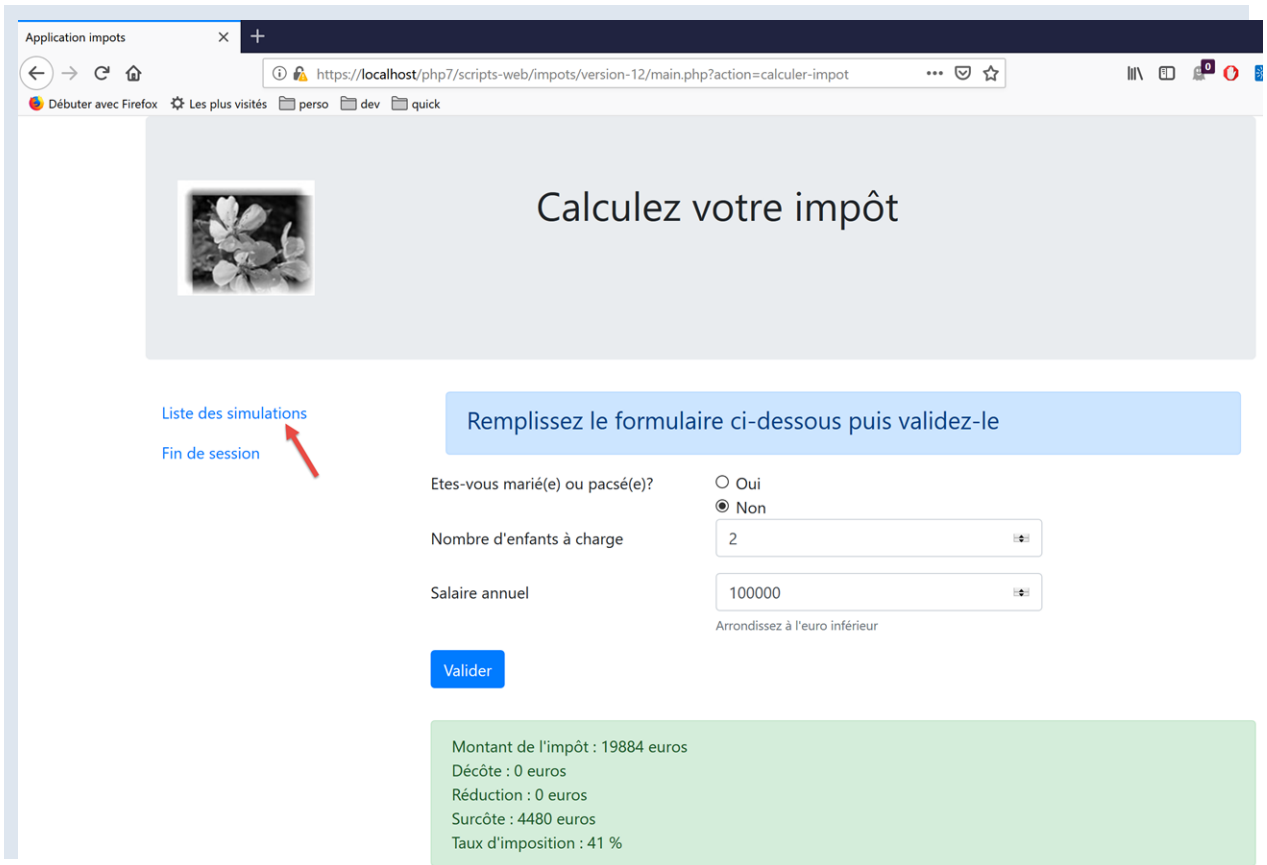
- en **[12-13]**, la nouvelle requête faite par le navigateur ;

Remplissons le formulaire que nous avons reçu ;

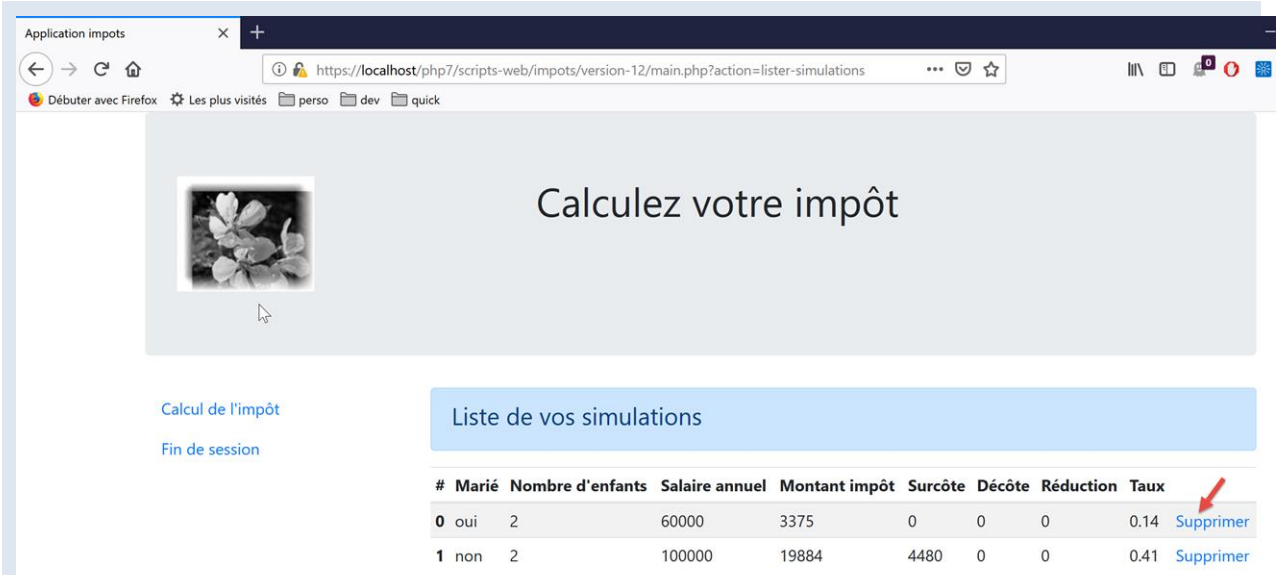


Puis faisons quelques simulations :

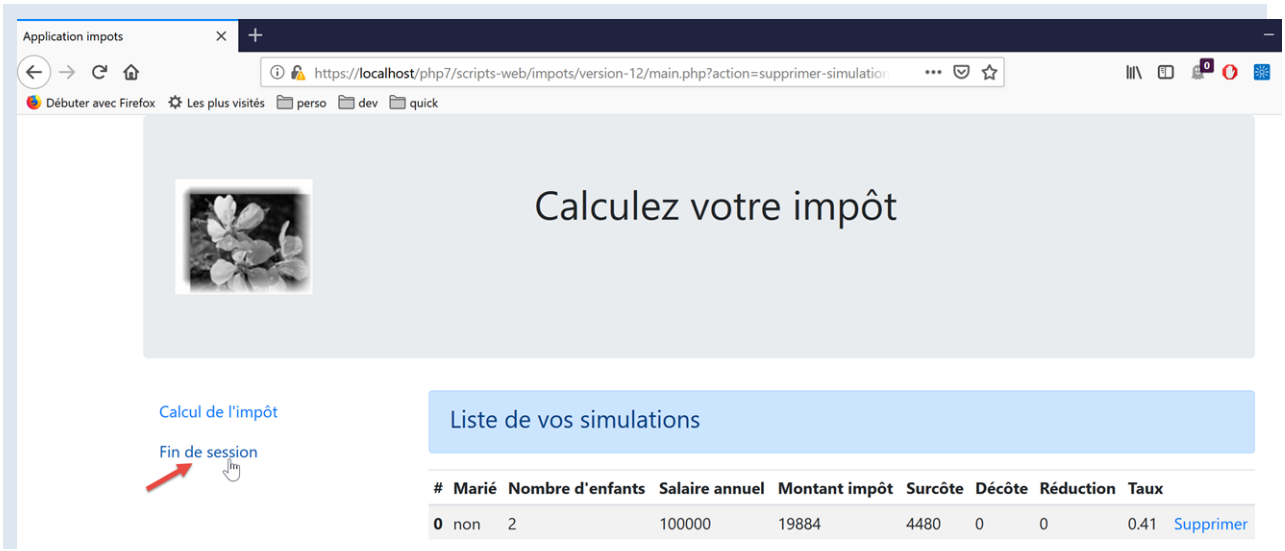

Application impots
https://localhost/php7/scripts-web/impots/version-12/main.php?action=calculer-impot
Débuter avec Firefox
Les plus visités
perso
dev
quick
Calculez votre impôt
Liste des simulations
Fin de session
Remplissez le formulaire ci-dessous puis validez-le
Etes-vous marié(e) ou pacsé(e)?
Oui
Non
Nombre d'enfants à charge
2
Salaire annuel
60000
Arrondissez à l'euro inférieur
Valider
Montant de l'impôt : 3375 euros
Décôte : 0 euros
Réduction : 0 euros
Surcôte : 0 euros
Taux d'imposition : 14 %


Application impots
https://localhost/php7/scripts-web/impots/version-12/main.php?action=calculer-impot
Débuter avec Firefox
Les plus visités
perso
dev
quick
Calculez votre impôt
Liste des simulations
Fin de session
Remplissez le formulaire ci-dessous puis validez-le
Etes-vous marié(e) ou pacsé(e)?
Oui
Non
Nombre d'enfants à charge
2
Salaire annuel
100000
Arrondissez à l'euro inférieur
Valider
Montant de l'impôt : 19884 euros
Décôte : 0 euros
Réduction : 0 euros
Surcôte : 4480 euros
Taux d'imposition : 41 %

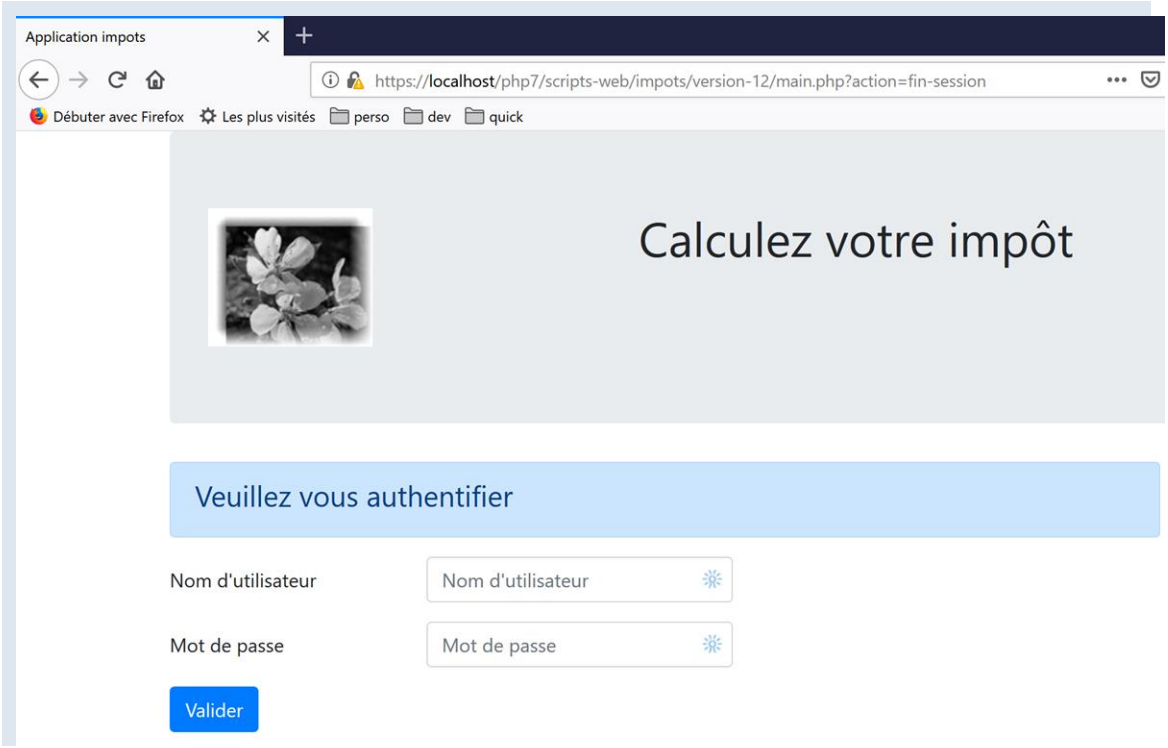
Demandons la liste des simulations :



Supprimons la 1re simulation :



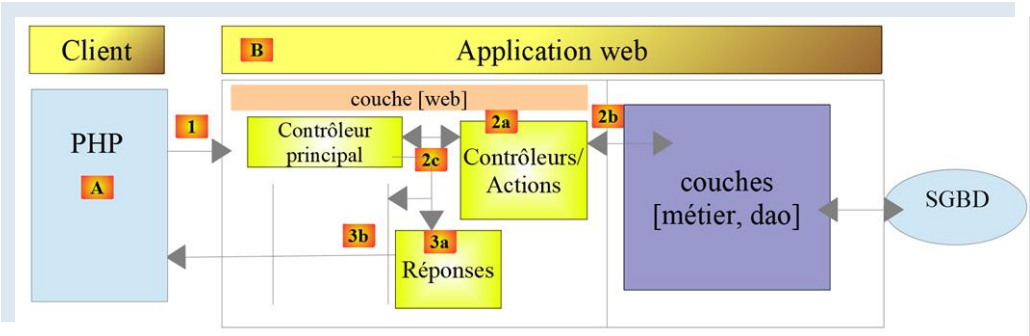
Terminons la session :



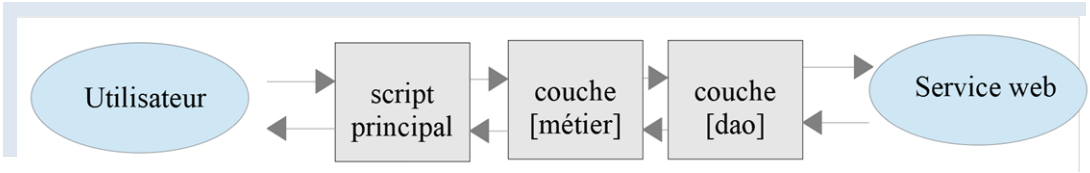
Le lecteur est invité à faire d'autres tests.

1.23.14 Client du service web jSON

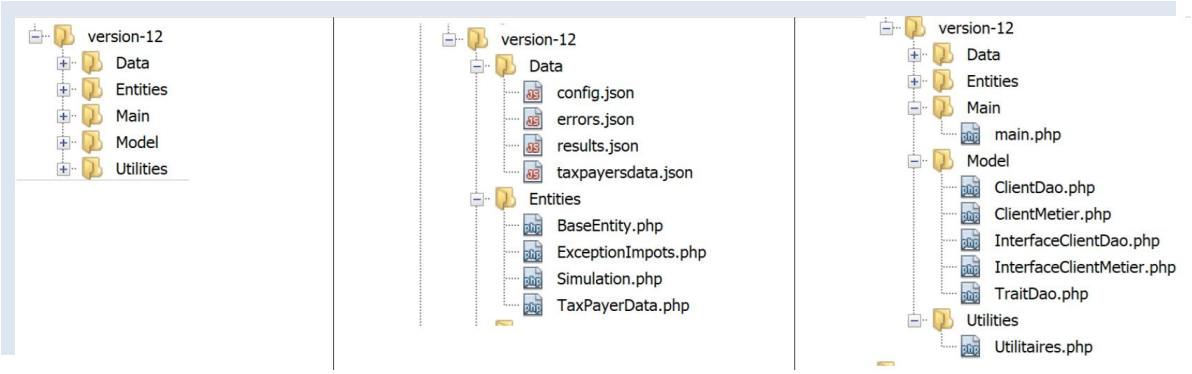
1.23.14.1 Architecture client / serveur



Nous nous intéressons maintenant au client jSON **[A]** du service web **[B]**. Le client **[A]**, comme le service web **[B]** a une structure en couches :



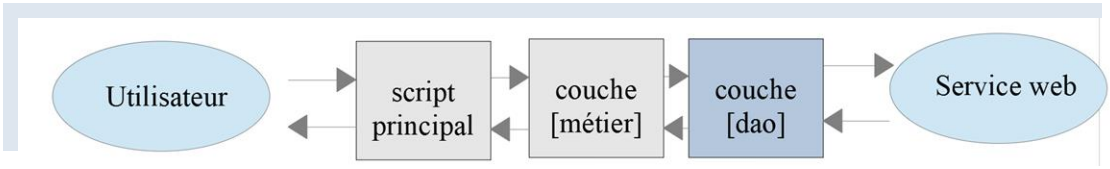
Cette architecture est reflétée par l'organisation du code suivante :



La plupart des classes ont déjà été rencontrées et expliquées :

BaseEntity	paragraphe lien.
TaxPayerData	paragraphe lien.
Simulation	paragraphe lien.
ExceptionImpots	paragraphe lien.
TraitDao	paragraphe lien.
Utilitaires	paragraphe lien.

1.23.14.2 La couche [dao]



1.23.14.2.1 Interface

L'interface de la couche **[dao]** sera la suivante **[InterfaceClientDao.php]** :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  interface InterfaceClientDao {
7.
8.    // lecture des données contribuables
9.    public function getTaxPayersData(string $taxPayersFilename, string $errorsFilename): array;
10.
11.   // calcul des impôts d'un contribuable
12.   public function calculerImpot(string $marié, int $enfants, int $salaire): Simulation;
13.
14.   // enregistrement des résultats
15.   public function saveResults(string $resultsFilename, array $simulations): void;
16.
17.   // authentification
18.   public function authentifierUtilisateur(String $user, string $password): void;
19.
20.   // liste des simulations
21.   public function listerSimulations(): array;
22.
23.   // supprimer une simulation
24.   public function supprimerSimulation(int $numéro): array;
25.
26.   // début de session
27.   public function initSession(string $type = 'json'): void;
28.
29.   // fin de session
```

```
30.   public function finSession(): void;
31. }
```

Commentaires

- ligne 9 : la méthode **[getTaxPayersData]** permet d'exploiter le fichier jSON des données des contribuables. Cette méthode est implémentée par le trait **[TraitDao]** déjà commenté (paragraphe lien) ;
- ligne 15 : la méthode **[saveResults]** permet de sauvegarder les résultats de plusieurs calculs d'impôt dans un fichier jSON. Là également, cette méthode est implémentée par le trait **[TraitDao]** déjà commenté (paragraphe lien) ;
- lignes 12, 18, 21, 27, 30 : on a créé une méthode pour chacune des actions acceptées par le service web ;

1.23.14.2.2 Implémentation

L'interface **[InterfaceClientDao]** est implémentée par la classe **[ClientDao]** suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances
6.  use Symfony\Component\HttpClient\HttpClient;
7.  use Symfony\Component\HttpClient\Response\CurlResponse;
8.
9.  class ClientDao implements InterfaceClientDao {
10.   // utilisation d'un Trait
11.   use TraitDao;
12.   // attributs
13.   private $urlServer;
14.   private $sessionCookie;
15.   private $verbose;
16.
17.   // constructeur
18.   public function __construct(string $urlServer, bool $verbose = TRUE) {
19.     $this->urlServer = $urlServer;
20.     $this->verbose = $verbose;
21.   }
22. …
23. }
```

Commentaires

- lignes 18-21 : le constructeur reçoit deux paramètres :
  - l'URL **[$urlServer]** du service web jSON ;
  - un booléen **[$verbose]** qui à TRUE indique que la classe doit afficher les réponses du serveur sur la console ;
- ligne 14 : le cookie de session. Le rôle de celui-ci a été décrit dans la version 09 du client (paragraphe lien) ;
- ligne 11 : la classe utilise le trait **[TraitDao]** qui implémente deux méthodes de l'interface :
  - **[getTaxPayersData(string $taxPayersFilename, string $errorsFilename): array]** ;
  - **[function calculerImpot(string $marié, int $enfants, int $salaire): Simulation]** ;

1.23.14.2.2.1 Méthode [initSession]

La méthode **[initSession]** est implémentée de la façon suivante :

```
1.  public function initSession(string $type = 'json'): void {
2.      // on crée un client HTTP
3.      $httpClient = HttpClient::create();
4.      // on fait la requête au serveur sans authentification
5.      $response = $httpClient->request('GET', $this->urlServer,
6.        ["query" => [
7.            "action" => "init-session",
8.            "type" => $type
9.          ],
10.         "verify_peer" => false
11.     ]);
12.     // on récupère la réponse
13.     $this->getResponse($response);
14.     // on récupère le cookie de session
15.     $headers = $response->getHeaders();
16.     if (isset($headers["set-cookie"])) {
17.       // cookie de session ?
18.       foreach ($headers["set-cookie"] as $cookie) {
19.         $match = [];
20.         $match = preg_match("/^PHPSESSID=(.+?);/", $cookie, $champs);
21.         if ($match) {
22.           $this->sessionCookie = "PHPSESSID=" . $champs[1];
```

```
23.         }
24.       }
25.     }
26. }
```

L'action **[init-session]** devant être la 1re action demandée au service web, la méthode **[initSession]** sera la 1re méthode de la couche **[dao]** à être appelée.

Commentaires

- ligne 1 : le type de session désirée est passé en paramètre. En l'absence de paramètre, ce sera une session jSON qui sera démarrée ;
- lignes 5-11 : une requête GET est faite au service web ;
- lignes 7-8 : les deux paramètres du GET ;
- ligne 10 : en cas d'échanges sécurisés (schéma https), le certificat de sécurité envoyé par le service web ne sera pas vérifié ;
- ligne 13 : la méthode **[getResponse]** récupère la réponse du serveur. Elle la rend sous forme d'un tableau. Ici, le résultat de la méthode n'est pas exploité. La méthode **[getResponse]** lance une exception si le code HTTP de la réponse du service web est différent de 200 OK ;
- lignes 14-25 : comme la méthode **[initSession]** est la 1re méthode de la couche **[dao]** à être exécutée, on récupère le cookie de session pour que les méthodes suivantes puissent le renvoyer au service web. Ce code a déjà été commenté dans la version 09 ;

1.23.14.2.2 La méthode [getResponse]

La méthode **[getResponse]** est chargée d'exploiter la réponse du service web :

```
1.  private function getResponse(CurlResponse $response) {
2.      // on récupère la réponse
3.      $json = $response->getContent(false);
4.      // logs
5.      if ($this->verbose) {
6.        print "$json\n";
7.      }
8.      // on récupère le statut de la réponse
9.      $statusCode = $response->getStatusCode();
10.     // erreur ?
11.     if ($statusCode !== 200) {
12.       // on a une erreur
13.       throw new ExceptionImpots($json);
14.     }
15.     // on rend sa réponse
16.     $array = json_decode($json, true);
17.     return $array["réponse"];
18. }
```

Commentaires

- ligne 1 : la méthode est privée ;
- ligne 1 : le paramètre de la méthode est la réponse du service web de type **[Symfony\Component\HttpClient\Response\CurlResponse]**, le type de réponse Symfony, lorsque **[HttpClient]** est implémenté par **[CurlClient]**, ç-à-d par la bibliothèque **[curl]** ;
- ligne 3 : on récupère la réponse jSON du serveur. On rappelle que le paramètre **[false]** est là pour empêcher Symfony de lancer une exception lorsque le statut de la réponse HTTP du serveur est dans le domaine **[3xx, 4xx, 5xx]** ;
- lignes 5-7 : si on est en mode **[$verbose]** alors on affiche la réponse du serveur sur la console ;
- lignes 9-14 : si le statut de la réponse HTTP du serveur est différent de 200, alors on lance une exception avec pour message d'erreur la réponse jSON du serveur ;
- ligne 16 : la chaîne jSON est décodée dans un tableau ;
- ligne 17 : les informations utiles sont dans **[$array["réponse"]]** ;

1.23.14.2.3 La méthode [authentifierUtilisateur]

La méthode **[authentifierUtilisateur]** est la suivante :

```
1.  public function authentifierUtilisateur(string $user, string $password): void {
2.      // on crée un client HTTP
3.      $httpClient = HttpClient::create();
4.      // on fait la requête au serveur avec authentification
5.      $response = $httpClient->request('POST', $this->urlServer,
6.        ["query" => [
7.            "action" => "authentifier-utilisateur"
8.          ],
9.          "body" => [
10.           "user" => $user,
11.           "password" => $password
```

```
12.         ],
13.         "verify_peer" => false,
14.         "headers" => ["Cookie" => $this->sessionCookie]
15.     ]);
16.     // on récupère la réponse
17.     $this->getResponse($response);
18. }
```

Commentaires

- ligne 5 : la requête du client est un POST ;
- lignes 6-8 : paramètres dans l'URL ;
- lignes 9-12 : paramètres du POST ;
- ligne 14 : le cookie de session ;
- ligne 17 : on lit la réponse. On sait qu'en cas d'erreur (code HTTP différent de 200), la méthode **[getResponse]** lance elle-même une exception ;

1.23.14.2.2.4 La méthode [calculerImpot]

```
1.  public function calculerImpot(string $marié, int $enfants, int $salaire): Simulation {
2.      // on crée un client HTTP
3.      $httpClient = HttpClient::create();
4.      // on fait la requête au serveur sans authentification mais avec le cookie de session
5.      $response = $httpClient->request('POST', $this->urlServer,
6.        ["query" => [
7.            "action" => "calculer-impot"],
8.          "body" => [
9.            "marié" => $marié,
10.           "enfants" => $enfants,
11.           "salaire" => $salaire
12.         ],
13.         "verify_peer" => false,
14.         "headers" => ["Cookie" => $this->sessionCookie]
15.     ]);
16.     // on récupère la réponse
17.     $array = $this->getResponse($response);
18.     return (new Simulation())->setFromArrayOfAttributes($array);
19. }
```

Commentaires

- ligne 6-7 : l'unique paramètre de l'URL ;
- lignes 8-12 : les trois paramètres du POST (ligne 5) ;
- ligne 17 : la réponse est exploitée ;
- ligne 18 : si on arrive là c'est que la méthode **[getResponse]** n'a pas lancé d'exception. On rend un objet **[Simulation]** initialisé avec le tableau rendu par **[getResponse]** ;

1.23.14.2.2.5 La méthode [listerSimulations]

```
1.  public function listerSimulations(): array {
2.      // on crée un client HTTP
3.      $httpClient = HttpClient::create();
4.      // on fait la requête au serveur sans authentification mais avec le cookie de session
5.      $response = $httpClient->request('GET', $this->urlServer,
6.        ["query" => [
7.            "action" => "lister-simulations"
8.          ],
9.          "verify_peer" => false,
10.         "headers" => ["Cookie" => $this->sessionCookie]
11.     ]);
12.     // on récupère la réponse
13.     return $this->getSimulations($response);
14. }
```

Commentaires

- ligne 5 : méthode GET ;
- lignes 6-8 : l'unique paramètre du GET ;
- ligne 13 : la récupération des simulations est confiée à la méthode privée **[getSimulations]** ;

1.23.14.2.2.6 La méthode [getSimulations]

```
1.  private function getSimulations(CurlResponse $response): array {
2.      // on récupère la réponse JSON
3.      $array = $this->getResponse($response);
```

```
4.      // on a un tableau d'bjets associatifs
5.      // on va en faire un tableau d'objets Simulation
6.      $simulations = [];
7.      foreach ($array as $simulation) {
8.        $simulations [] = (new Simulation())->setFromArrayOfAttributes($simulation);
9.      }
10.     // on rend la liste d'objets Simulation
11.     return $simulations;
12. }
```

Commentaires

- ligne 3 : on récupère le tableau issu de la réponse. C'est un tableau de tableaux, chacun de ces derniers ayant tous les attributs d'un objet **[Simulation]** ;
- ligne 6 : si on arrive là, c'est que la méthode **[getResponse]** n'a pas lancé d'exception ;
- lignes 6-9 : on exploite la réponse pour construire un tableau d'objets **[Simulation]** ;
- ligne 11 : on rend ce tableau ;

1.23.14.2.2.7 La méthode [SupprimerSimulation]

```
1.  public function supprimerSimulation(int $numéro): array {
2.      // on crée un client HTTP
3.      $httpClient = HttpClient::create();
4.      // on fait la requête au serveur sans authentification mais avec le cookie de session
5.      $response = $httpClient->request('GET', $this->urlServer,
6.        ["query" => [
7.            "action" => "supprimer-simulation",
8.            "numéro" => $numéro
9.          ],
10.         "verify_peer" => false,
11.         "headers" => ["Cookie" => $this->sessionCookie]
12.     ]);
13.     // on récupère la réponse
14.     return $this->getSimulations($response);
15. }
```

Commentaires

- ligne 5 : on fait une requête GET ;
- lignes 6-9 : les deux paramètres de l'URL ;
- ligne 14 : après une suppression, le serveur rend le nouveau tableau des simulations. On rend ce tableau ;

1.23.14.2.2.8 La méthode [finSession]

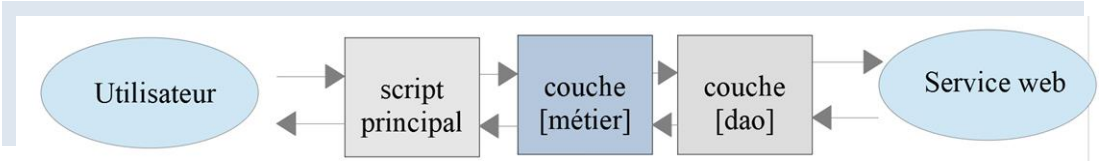
Une session de travail avec le service web se termine normalement par l'appel à la méthode **[finSession]** :

```
1.  public function finSession(): void {
2.      // on crée un client HTTP
3.      $httpClient = HttpClient::create();
4.      // on fait la requête au serveur sans authentification mais avec le cookie de session
5.      $response = $httpClient->request('GET', $this->urlServer,
6.        ["query" => [
7.            "action" => "fin-session"
8.          ],
9.          "verify_peer" => false,
10.         "headers" => ["Cookie" => $this->sessionCookie]
11.     ]);
12.     // on récupère la réponse
13.     $this->getResponse($response);
14. }
```

Commentaires

- ligne 5 : on fait une requête GET ;
- lignes 6-8 : l'unique paramètre de l'URL ;
- ligne 13 : on lit la réponse. Une exception sera lancée si le code HTTP de la réponse est différent de 200 ;

1.23.14.3 La couche [métier]



1.23.14.3.1 L'interface

L'interface de la couche **[métier]** est la suivante **[InterfaceClientMetier.php]** :

```
1.  <?php
2.
3.  // espace de noms
4.  namespace Application;
5.
6.  interface InterfaceClientMetier {
7.
8.    // calcul des impôts d'un contribuable
9.    public function calculerImpot(string $marié, int $enfants, int $salaire): Simulation;
10.
11.   // calcul des impôts en mode batch
12.   public function executeBatchImpots(string $taxPayersFileName, string $resultsFilename, string $errorsFileName): void;
13.
14.   // authentification
15.   public function authentifierUtilisateur(String $user, string $password): void;
16.
17.   // liste des simulations
18.   public function listerSimulations(): array;
19.
20.   // enregistrement des résultats
21.   public function saveResults(string $resultsFilename, array $simulations): void;
22.
23.   // supprimer une simulation
24.   public function supprimerSimulation(int $numéro): array;
25.
26.   // début de session
27.   public function initSession(string $type = 'json'): void;
28.
29.   // fin de session
30.   public function finSession(): void;
31. }
```

Commentaires

- seule la méthode **[executeBatchImpots]** de la ligne 12 est spécifique à la couche **[métier]**. Toutes les autres appartiennent à la couche **[dao]** qui les implémente ;

1.23.14.3.2 La classe [ClientMetier]

La classe implémentant la couche **[métier]** est la suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  class ClientMetier implements InterfaceClientMetier {
6.    // attribut
7.    private $clientDao;
8.
9.    // constructeur
10.   public function __construct(InterfaceClientDao $clientDao) {
11.     $this->clientDao = $clientDao;
12.   }
13.
14.   // calcul de l'impôt
15.   public function calculerImpot(string $marié, int $enfants, int $salaire): Simulation {
16.     return $this->clientDao->calculerImpot($marié, $enfants, $salaire);
17.   }
18.
19.   // calcul des impôts en mode batch
20.   public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string
    $errorsFileName): void {
21.     // on laisse remonter les exceptions qui proviennent de la couche [dao]
22.     // on récupère les données contribuables
23.     $taxPayersData = $this->clientDao->getTaxPayersData($taxPayersFileName, $errorsFileName);
```

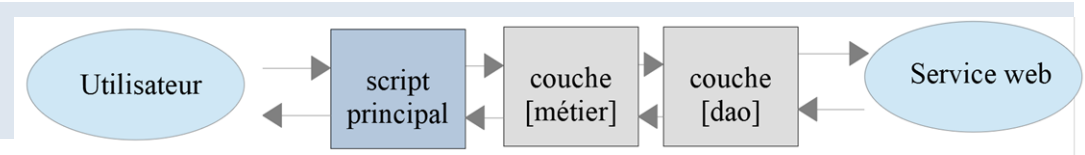
```
24.     // tableau des résultats
25.     $simulations = [];
26.     // on les exploite
27.     foreach ($taxPayersData as $taxPayerData) {
28.       // on calcule l'impôt
29.       $simulations [] = $this->calculerImpot(
30.         $taxPayerData->getMarié(),
31.         $taxPayerData->getEnfants(),
32.         $taxPayerData->getSalaire());
33.     }
34.     // enregistrement des résultats
35.     if ($resultsFileName !== NULL) {
36.       $this->clientDao->saveResults($resultsFileName, $simulations);
37.     }
38.   }
39.
40.   public function authentifierUtilisateur(String $user, string $password): void {
41.     $this->clientDao->authentifierUtilisateur($user, $password);
42.   }
43.
44.   public function listerSimulations(): array {
45.     return $this->clientDao->listerSimulations();
46.   }
47.
48.   public function saveResults(string $resultsFilename, array $simulations): void {
49.     $this->clientDao->saveResults($resultsFilename, $simulations);
50.   }
51.
52.   public function supprimerSimulation(int $numéro): array {
53.     return $this->clientDao->supprimerSimulation($numéro);
54.   }
55.
56.   public function finSession(): void {
57.     $this->clientDao->finSession();
58.   }
59.
60.   public function initSession(string $type = 'json'): void {
61.     $this->clientDao->initSession($type);
62.   }
63.
64. }
```

Commentaires

- lignes 10-12 : pour se construire, la couche **[métier]** a besoin d'une référence sur la couche **[dao]** ;
- lignes 20-38 : seule la méthode **[executeBatchImpots]** est spécifique à la couche **[métier]**. L'implémentation des autres méthodes délègue le travail à faire aux méthodes de mêmes noms dans la couche **[dao]** ;
- ligne 23 : on s'adresse à la couche **[dao]** pour obtenir dans un tableau d'objets de type **[TaxPayerData]** les données des contribuables ;
- ligne 25 : on va cumuler dans le tableau **[$simulations]** les différentes simulations calculées ;
- lignes 27-33 : on calcule l'impôt de chacun des contribuables du tableau **[$taxPayersData]** ;
- lignes 35-37 : les résultats obtenus dans le tableau **[$simulations]** sont sauvegardés dans un fichier jSON ;

Note : La couche **[métier]** ne fait quasiment rien. On pourrait décider de la supprimer et de tout rassembler dans la couche **[dao]**.

1.23.14.4 Le script principal



Le script principal est configuré par le fichier **[config.json]** suivant :

```
1.  {
2.      "taxPayersDataFileName": "Data/taxpayersdata.json",
3.      "resultsFileName": "Data/results.json",
4.      "errorsFileName": "Data/errors.json",
5.      "rootDirectory": "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-12",
```

```
    "dependencies": [
        "/Entities/BaseEntity.php",
        "/Entities/TaxPayerData.php",
        "/Entities/Simulation.php",
        "/Entities/ExceptionImpots.php",
        "/Utilities/Utilitaires.php",
        "/Model/InterfaceClientDao.php",
        "/Model/TraitDao.php",
        "/Model/ClientDao.php",
        "/Model/InterfaceClientMetier.php",
        "/Model/ClientMetier.php"
    ],
    "absoluteDependencies": [
        "C:/myprograms/laragon-lite/www/vendor/autoload.php"
    ],
    "user": {
        "login": "admin",
        "passwd": "admin"
    },
    "urlServer": "https://localhost:443/php7/scripts-web/impots/version-12/main.php"
}
```

Le script principal **[main.php]** est le suivant :

```
<?php

// respect strict des types déclarés des paramètres de foctions
declare(strict_types = 1);

// espace de noms
namespace Application;

// gestion des erreurs par PHP
// ini_set("display_errors", "0");
//
// chemin du fichier de configuration
define("CONFIG_FILENAME", "../Data/config.json");

// on récupère la configuration
$config = \json_decode(file_get_contents(CONFIG_FILENAME), true);

// on inclut les dépendances nécessaires au script
$rootDirectory = $config["rootDirectory"];
foreach ($config["dependencies"] as $dependency) {
  require "$rootDirectory/$dependency";
}
// dépendances absolues (bibliothèques tierces)
foreach ($config["absoluteDependencies"] as $dependency) {
  require "$dependency";
}

// définition des constantes
define("TAXPAYERSDATA_FILENAME", "$rootDirectory/{$config["taxPayersDataFileName"]}");
define("RESULTS_FILENAME", "$rootDirectory/{$config["resultsFileName"]}");
define("ERRORS_FILENAME", "$rootDirectory/{$config["errorsFileName"]}");
//
// dépendances Symfony
use Symfony\Component\HttpClient\HttpClient;

// création de la couche [dao]
$clientDao = new ClientDao($config["urlServer"]);
// création de la couche [métier]
$clientMetier = new ClientMetier($clientDao);

// calcul de l'impôts en mode batch
try {
  // initialisation de la session
  $clientMetier->initSession('json');
  // authentification
  $clientMetier->authentifierUtilisateur($config["user"]["login"], $config["user"]["passwd"]);
  // calcul d'impots sans sauvegarde des résultats
  $clientMetier->executeBatchImpots(TAXPAYERSDATA_FILENAME, NULL, ERRORS_FILENAME);
  // liste des simulations
  $clientMetier->listerSimulations();
  // suppression d'une simulation
  $simulations = $clientMetier->supprimerSimulation(1);
  // sauvegarde des résultats
```

```
54.   $clientMetier->saveResults(RESULTS_FILENAME, $simulations);
55.   // fin de session
56.   $clientMetier->finSession();
57.   // action sans être authentifié - doit planter
58.   $clientMetier->listerSimulations();
59. } catch (ExceptionImpots $ex) {
60.   // on affiche l'erreur
61.   print "Une erreur s'est produite : " . $ex->getMessage() . "\n";
62. }
63. // fin
64. print "Terminé\n";
65. exit();
```

Commentaires

- lignes 12-16 : exploitation du fichier de configuration **[config.json]** ;
- lignes 18-26 : chargement de toutes les dépendances ;
- lignes 28-34 : définition de constantes et d'alias ;
- lignes 36-39 : construction des couches **[dao]** et **[métier]** ;
- ligne 44 : initialisation d'une session jSON ;
- ligne 46 : on s'authentifie auprès du serveur ;
- ligne 48 : calcul de l'impôt d'une série de contribuables. On ne sauvegarde pas les résultats (2e paramètre NULL) ;
- ligne 50 : on demande les résultats de tous ces calculs ;
- ligne 52 : on supprime la simulation n° 1 (la 2e de la liste) ;
- ligne 54 : on sauvegarde les simulations qui restent ;
- ligne 56 : on termine la session. Cela signifie que le cookie de session est détruit ;
- ligne 58 : on demande la liste des simulations. Comme le cookie de session a été détruit, l'authentification doit être refaite. On doit donc avoir une exception disant qu'on n'est pas authentifié ;

Le fichier **[taxpayersdata.json]** est le suivant :

```
1.  [
2.      {
3.          "marié": "oui",
4.          "enfants": 2,
5.          "salaire": 55555
6.      },
7.      {
8.          "marié": "ouix",
9.          "enfants": "2x",
10.         "salaire": "55555x"
11.     },
12.     {
13.         "marié": "oui",
14.         "enfants": "2",
15.         "salaire": 50000
16.     },
17.     {
18.         "marié": "oui",
19.         "enfants": 3,
20.         "salaire": 50000
21.     },
22.     {
23.         "marié": "non",
24.         "enfants": 2,
25.         "salaire": 100000
26.     },
27.     {
28.         "marié": "non",
29.         "enfants": 3,
30.         "salaire": 100000
31.     },
32.     {
33.         "marié": "oui",
34.         "enfants": 3,
35.         "salaire": 100000
36.     },
37.     {
38.         "marié": "oui",
39.         "enfants": 5,
40.         "salaire": 100000
41.     },
42.     {
```

```
        "marié": "non",
        "enfants": 0,
        "salaire": 100000
    },
    {
        "marié": "oui",
        "enfants": 2,
        "salaire": 30000
    },
    {
        "marié": "non",
        "enfants": 0,
        "salaire": 200000
    },
    {
        "marié": "oui",
        "enfants": 3,
        "salaire": 20000
    }
]
```

Il y a 12 contribuables dont 1 est incorrect. Cela fait donc 11 simulations au total. L'une d'elles va être supprimée. Il doit en rester 10.

Après exécution du script principal, le fichier jSON **`[results.json]`** est le suivant :

```
[
        {
                "marié": "oui",
                "enfants": "2",
                "salaire": "55555",
                "impôt": 2814,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.14
        },
        {
                "marié": "oui",
                "enfants": "3",
                "salaire": "50000",
                "impôt": 0,
                "surcôte": 0,
                "décôte": 720,
                "réduction": 0,
                "taux": 0.14
        },
        {
                "marié": "non",
                "enfants": "2",
                "salaire": "100000",
                "impôt": 19884,
                "surcôte": 4480,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.41
        },
        {
                "marié": "non",
                "enfants": "3",
                "salaire": "100000",
                "impôt": 16782,
                "surcôte": 7176,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.41
        },
        {
                "marié": "oui",
                "enfants": "3",
                "salaire": "100000",
                "impôt": 9200,
                "surcôte": 2180,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.3
```

```
        },
        {
                "marié": "oui",
                "enfants": "5",
                "salaire": "100000",
                "impôt": 4230,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.14
        },
        {
                "marié": "non",
                "enfants": "0",
                "salaire": "100000",
                "impôt": 22986,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.41
        },
        {
                "marié": "oui",
                "enfants": "2",
                "salaire": "30000",
                "impôt": 0,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0
        },
        {
                "marié": "non",
                "enfants": "0",
                "salaire": "200000",
                "impôt": 64210,
                "surcôte": 7498,
                "décôte": 0,
                "réduction": 0,
                "taux": 0.45
        },
        {
                "marié": "oui",
                "enfants": "3",
                "salaire": "20000",
                "impôt": 0,
                "surcôte": 0,
                "décôte": 0,
                "réduction": 0,
                "taux": 0
        }
]
```

Il y a bien 10 simulations.

Le fichier jSON **[errors.json]** a le contenu suivant :

```
{
        "numéro": 1,
        "erreurs": [
                {
                        "marié": "ouix"
                },
                {
                        "enfants": "2x"
                },
                {
                        "salaire": "55555x"
                }
        ]
}
```

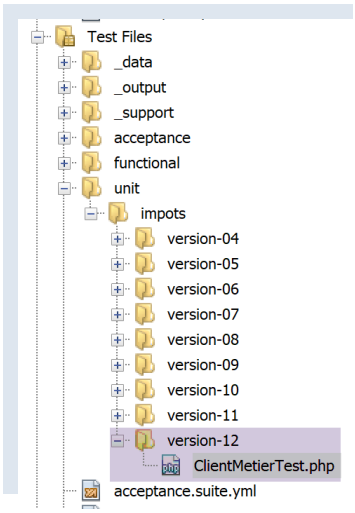
Les résultats console sont les suivants (en mode verbose, les réponses jSON du serveur sont affichées sur la console) :

```
{"action":"init-session","état":700,"réponse":"session démarrée avec type [json]"}
{"action":"authentifier-utilisateur","état":200,"réponse":"Authentification réussie [admin, admin]"}
```

```
3.  {"action":"calculer-
    impot","état":300,"réponse":{"marié":"oui","enfants":"2","salaire":"55555","impôt":2814,"surcôte":0,"décôte
    ":0,"réduction":0,"taux":0.14}}
4.  {"action":"calculer-
    impot","état":300,"réponse":{"marié":"oui","enfants":"2","salaire":"50000","impôt":1384,"surcôte":0,"décôte
    ":384,"réduction":347,"taux":0.14}}
5.  {"action":"calculer-
    impot","état":300,"réponse":{"marié":"oui","enfants":"3","salaire":"50000","impôt":0,"surcôte":0,"décôte":7
    20,"réduction":0,"taux":0.14}}
6.  {"action":"calculer-
    impot","état":300,"réponse":{"marié":"non","enfants":"2","salaire":"100000","impôt":19884,"surcôte":4480,"d
    écôte":0,"réduction":0,"taux":0.41}}
7.  {"action":"calculer-
    impot","état":300,"réponse":{"marié":"non","enfants":"3","salaire":"100000","impôt":16782,"surcôte":7176,"d
    écôte":0,"réduction":0,"taux":0.41}}
8.  {"action":"calculer-
    impot","état":300,"réponse":{"marié":"oui","enfants":"3","salaire":"100000","impôt":9200,"surcôte":2180,"dé
    côte":0,"réduction":0,"taux":0.3}}
9.  {"action":"calculer-
    impot","état":300,"réponse":{"marié":"oui","enfants":"5","salaire":"100000","impôt":4230,"surcôte":0,"décôt
    e":0,"réduction":0,"taux":0.14}}
10. {"action":"calculer-
    impot","état":300,"réponse":{"marié":"non","enfants":"0","salaire":"100000","impôt":22986,"surcôte":0,"décô
    te":0,"réduction":0,"taux":0.41}}
11. {"action":"calculer-
    impot","état":300,"réponse":{"marié":"oui","enfants":"2","salaire":"30000","impôt":0,"surcôte":0,"décôte":0
    ,"réduction":0,"taux":0}}
12. {"action":"calculer-
    impot","état":300,"réponse":{"marié":"non","enfants":"0","salaire":"200000","impôt":64210,"surcôte":7498,"d
    écôte":0,"réduction":0,"taux":0.45}}
13. {"action":"calculer-
    impot","état":300,"réponse":{"marié":"oui","enfants":"3","salaire":"20000","impôt":0,"surcôte":0,"décôte":0
    ,"réduction":0,"taux":0}}
14. {"action":"lister-
    simulations","état":500,"réponse":[{"marié":"oui","enfants":"2","salaire":"55555","impôt":2814,"surcôte":0,
    "décôte":0,"réduction":0,"taux":0.14,"arrayOfAttributes":null},{"marié":"oui","enfants":"2","salaire":"5000
    0","impôt":1384,"surcôte":0,"décôte":384,"réduction":347,"taux":0.14,"arrayOfAttributes":null},{"marié":"ou
    i","enfants":"3","salaire":"50000","impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14,"arrayOfAtt
    ributes":null},{"marié":"non","enfants":"2","salaire":"100000","impôt":19884,"surcôte":4480,"décôte":0,"réd
    uction":0,"taux":0.41,"arrayOfAttributes":null},{"marié":"non","enfants":"3","salaire":"100000","impôt":167
    82,"surcôte":7176,"décôte":0,"réduction":0,"taux":0.41,"arrayOfAttributes":null},{"marié":"oui","enfants":"
    3","salaire":"100000","impôt":9200,"surcôte":2180,"décôte":0,"réduction":0,"taux":0.3,"arrayOfAttributes":n
    ull},{"marié":"oui","enfants":"5","salaire":"100000","impôt":4230,"surcôte":0,"décôte":0,"réduction":0,"tau
    x":0.14,"arrayOfAttributes":null},{"marié":"non","enfants":"0","salaire":"100000","impôt":22986,"surcôte":0
    ,"décôte":0,"réduction":0,"taux":0.41,"arrayOfAttributes":null},{"marié":"oui","enfants":"2","salaire":"300
    00","impôt":0,"surcôte":0,"décôte":0,"réduction":0,"taux":0,"arrayOfAttributes":null},{"marié":"non","enfan
    ts":"0","salaire":"200000","impôt":64210,"surcôte":7498,"décôte":0,"réduction":0,"taux":0.45,"arrayOfAttrib
    utes":null},{"marié":"oui","enfants":"3","salaire":"20000","impôt":0,"surcôte":0,"décôte":0,"réduction":0,"
    taux":0,"arrayOfAttributes":null}]}
15. {"action":"supprimer-
    simulation","état":600,"réponse":[{"marié":"oui","enfants":"2","salaire":"55555","impôt":2814,"surcôte":0,"
    décôte":0,"réduction":0,"taux":0.14,"arrayOfAttributes":null},{"marié":"oui","enfants":"3","salaire":"50000
    ","impôt":0,"surcôte":0,"décôte":720,"réduction":0,"taux":0.14,"arrayOfAttributes":null},{"marié":"non","en
    fants":"2","salaire":"100000","impôt":19884,"surcôte":4480,"décôte":0,"réduction":0,"taux":0.41,"arrayOfAtt
    ributes":null},{"marié":"non","enfants":"3","salaire":"100000","impôt":16782,"surcôte":7176,"décôte":0,"réd
    uction":0,"taux":0.41,"arrayOfAttributes":null},{"marié":"oui","enfants":"3","salaire":"100000","impôt":920
    0,"surcôte":2180,"décôte":0,"réduction":0,"taux":0.3,"arrayOfAttributes":null},{"marié":"oui","enfants":"5"
    ,"salaire":"100000","impôt":4230,"surcôte":0,"décôte":0,"réduction":0,"taux":0.14,"arrayOfAttributes":null}
    ,{"marié":"non","enfants":"0","salaire":"100000","impôt":22986,"surcôte":0,"décôte":0,"réduction":0,"taux":
    0.41,"arrayOfAttributes":null},{"marié":"oui","enfants":"2","salaire":"30000","impôt":0,"surcôte":0,"décôte
    ":0,"réduction":0,"taux":0,"arrayOfAttributes":null},{"marié":"non","enfants":"0","salaire":"200000","impôt
    ":64210,"surcôte":7498,"décôte":0,"réduction":0,"taux":0.45,"arrayOfAttributes":null},{"marié":"oui","enfan
    ts":"3","salaire":"20000","impôt":0,"surcôte":0,"décôte":0,"réduction":0,"taux":0,"arrayOfAttributes":null}
    ]}
16. {"action":"fin-session","état":400,"réponse":"session supprimée"}
17. {"action":"lister-simulations","état":103,"réponse":["pas de session en cours. Commencer par action [init-
    session]"]}
18. Une erreur s'est produite : {"action":"lister-simulations","état":103,"réponse":["pas de session en cours.
    Commencer par action [init-session]"]}
19. Terminé
```

1.23.14.5 Tests [Codeception]

Comme pour les précédents clients, le client de la version 12 peut faire l'objet de tests **[Codeception]** :



Le code de la classe de test de la couche **[métier]** du client est analogue à celui des classes de test des précédents clients :

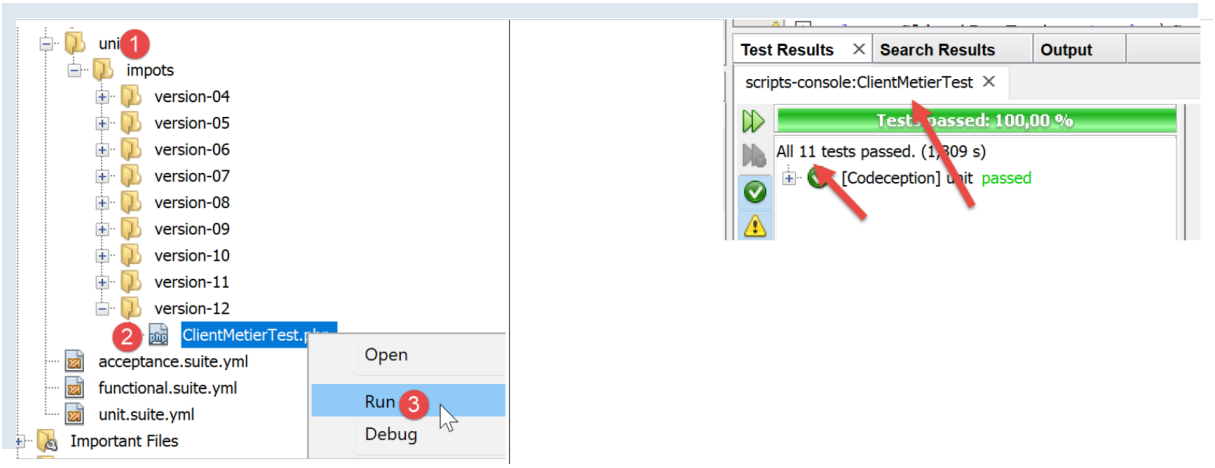
```
1.  <?php
2.
3.  // respect strict des types déclarés des paramètres de foctions
4.  declare (strict_types=1);
5.
6.  // espace de noms
7.  namespace Application;
8.
9.  // définition des constantes
10. define("ROOT", "C:/Data/st-2019/dev/php7/poly/scripts-console/impots/version-12");
11. // chemin du fichier de configuration
12. define("CONFIG_FILENAME", ROOT . "/Data/config.json");
13.
14. // on récupère la configuration
15. $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
16.
17. // on inclut les dépendances nécessaires au script
18. $rootDirectory = $config["rootDirectory"];
19. foreach ($config["dependencies"] as $dependency) {
20.   require "$rootDirectory$dependency";
21. }
22. // dépendances absolues (bibliothèques tierces)
23. foreach ($config["absoluteDependencies"] as $dependency) {
24.   require "$dependency";
25. }
26. // dépendances Symfony
27. use Symfony\Component\HttpClient\HttpClient;
28.
29. // classe de test
30. class ClientDaoTest extends \Codeception\Test\Unit {
31.   // couche dao
32.   private $clientDao;
33.
34.   public function __construct() {
35.     parent::__construct();
36.     // on récupère la configuration
37.     $config = \json_decode(\file_get_contents(CONFIG_FILENAME), true);
38.     // création de la couche [dao]
39.     $clientDao = new ClientDao($config["urlServer"]);
40.     // création de la couche [métier]
41.     $this->métier = new ClientMetier($clientDao);
42.     // initialisation session
43.     $this->métier->initSession("json");
44.     // authentification
45.     $this->métier->authentifierUtilisateur("admin", "admin");
46.   }
47.
48.   // tests
49.   public function test1() {
50.     $simulation = $this->métier->calculerImpot("oui", 2, 55555);
```

```
51.     $this->assertEqualsWithDelta(2815, $simulation->getImpôt(), 1);
52.     $this->assertEqualsWithDelta(0, $simulation->getSurcôte(), 1);
53.     $this->assertEqualsWithDelta(0, $simulation->getDécôte(), 1);
54.     $this->assertEqualsWithDelta(0, $simulation->getRéduction(), 1);
55.     $this->assertEquals(0.14, $simulation->getTaux());
56.   }
57.
58.   public function test2() {
59.     ….
60.   }
61.
62.   …
63.   public function test11() {
64.     …
65.   }
66.
67. }
```

Commentaires

- lignes 34-46 : on rappelle que le constructeur de la classe de test est exécuté avant chaque test ;
- lignes 38-41 : construction des couches **[dao]** et **[métier]** ;
- lignes 42-45 : les méthodes de test **[test1…, test11]** testent la méthode **[calculerImpot]**. Pour que cela soit possible, il faut auparavant initialiser une session jSON et s'authentifier ;

Les résultats du test sont les suivants :



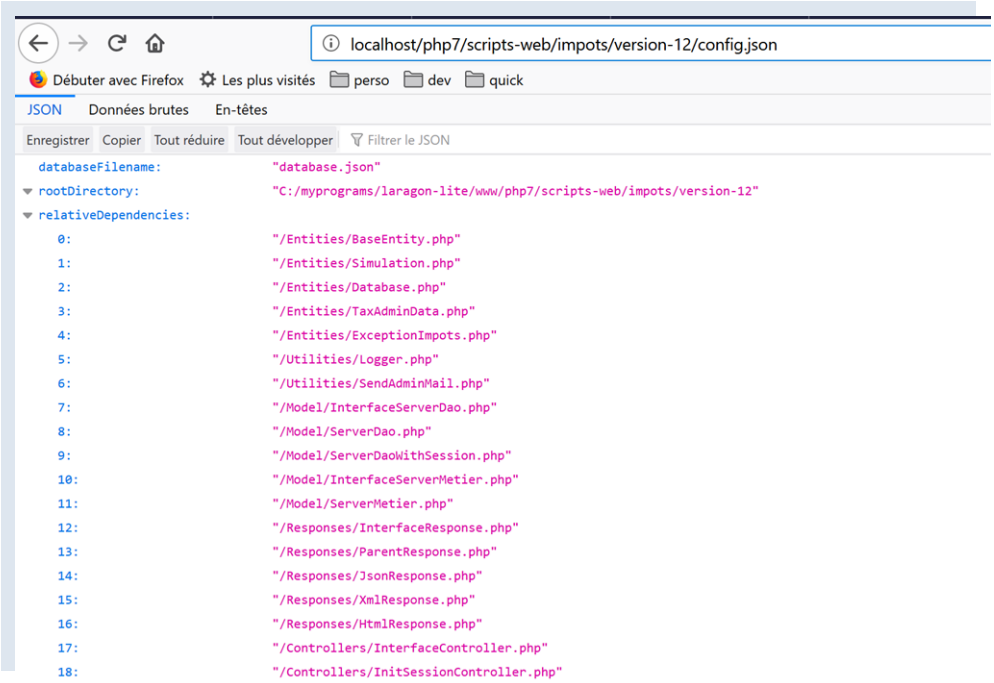
Beaucoup d'autres tests devraient être faits :

- tester les différentes méthodes de la couche **[dao]** ;
- tester les états rendus par le serveur web. Ces états sont importants puisque leur valeur décide de la page HTML à afficher ;

1.24 Exercice d'application – version 13

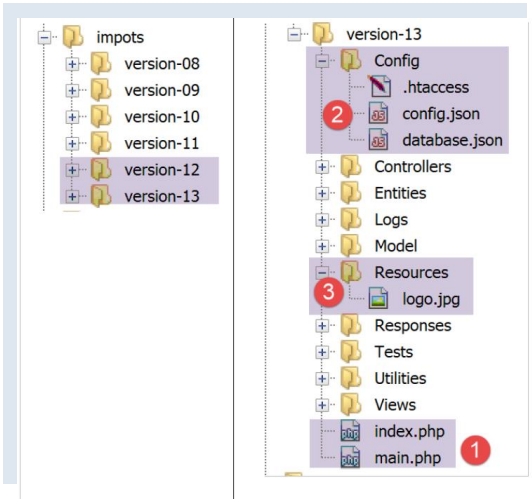
La version 13 amène peu de changements : elle sécurise l'accès aux fichiers de l'application.

Avec la version 12, on peut demander l'URL suivante **[http://localhost/php7/scripts-web/impots/version-12/config.json]**. On obtient alors la page suivante (Firefox) :



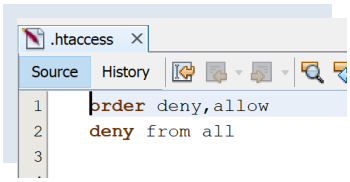
Or ce fichier **[config.json]** contient des informations sensibles tels les identifiants des utilisateurs autorisés à utiliser l'application. Il ne faut pas qu'il soit accessible aux utilisateurs. Il en est de même pour tous les fichiers de l'application à l'exception des fichiers **[main.php, index.php, Views/logo.jpg]** qui eux doivent être accessibles de l'extérieur. En effet les vues ont besoin d'avoir un accès HTTP au logo de l'application. La version 13 amène une solution simple à ce problème.

Dans Netbeans, nous faisons un copier-coller du dossier **[version-12]** dans **[version-13]** :



- en **[1]**, dans le dossier racine de l'application, nous ne laissons que les scripts **[index.php, main.php]** ;
- en **[2]**, les fichiers de configuration sont placés dans un dossier **[Config]** ;
- en **[3]**, l'image **[logo.jpg]** est placée dans un dossier **[Resources]** ;

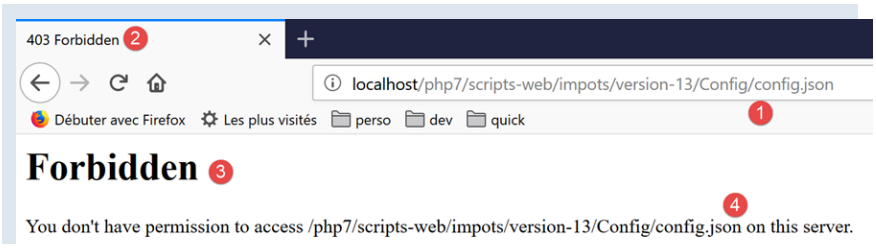
Le serveur HTTP utilisé est ici un serveur Apache. Celui-ci permet de contrôler l'accès à un dossier via un fichier **[.htaccess]**. Dans tous les dossiers auxquels nous voulons interdire un accès direct par URL, nous créons le fichier **[.htaccess]** suivant :

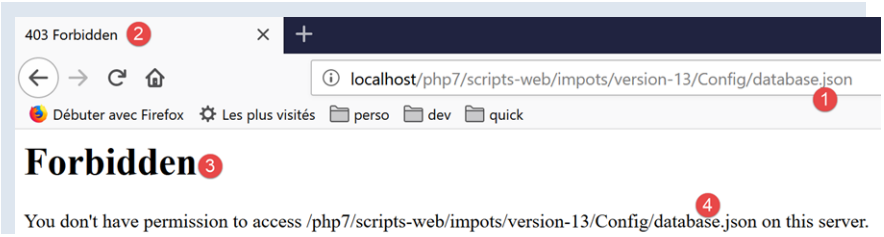


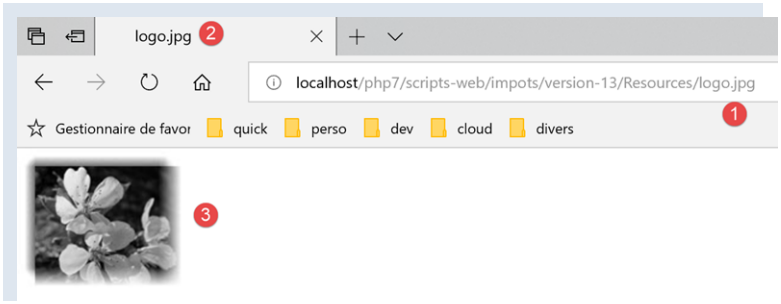
Ces deux lignes font que tout accès au dossier est interdit à tous.

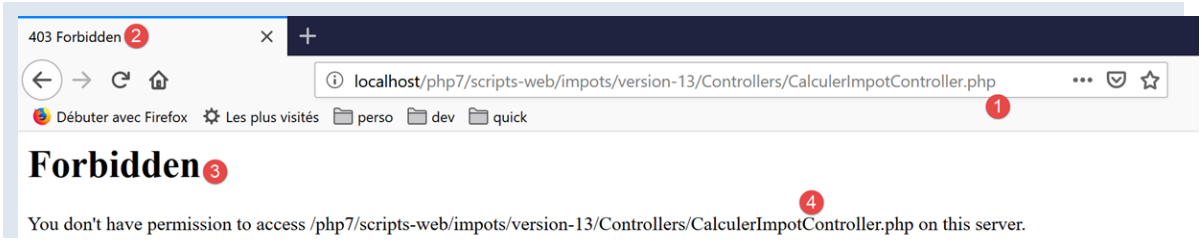
Nous plaçons ce fichier dans tous les dossiers de l'application sauf dans le dossier racine et le dossier **[Resources]**. Finalement seuls trois fichiers sont accessibles de l'extérieur : **[index.php, main.php, Resources/logo.jpg]**.

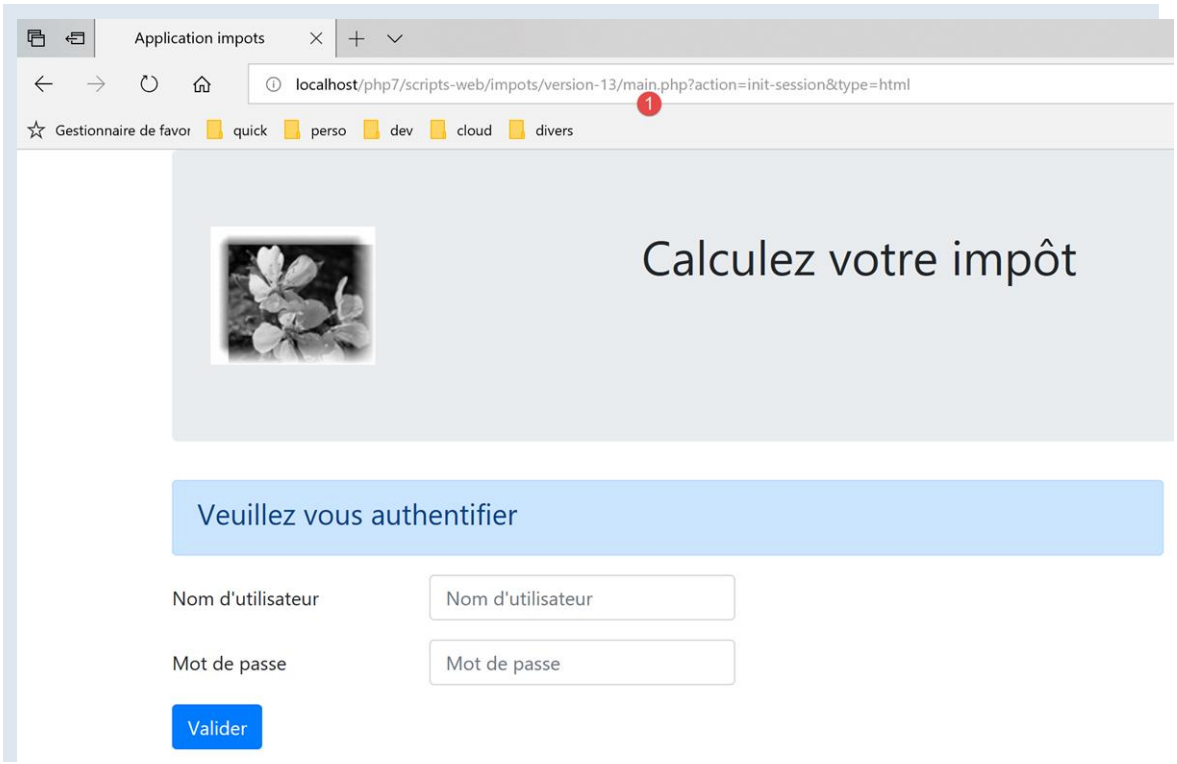
Faisons quelques essais :





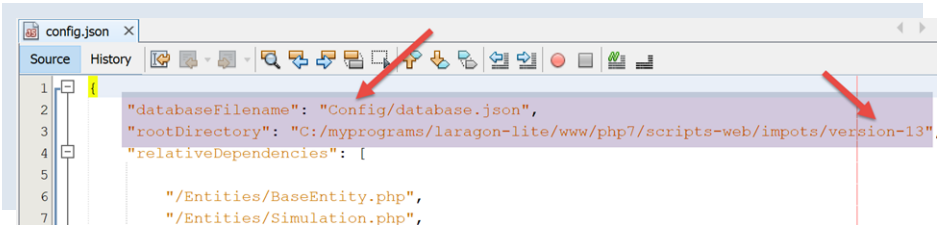




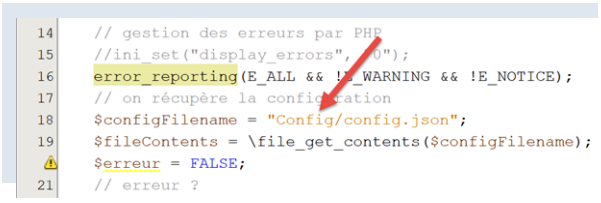


Quelques modifications doivent être faites dans le code :

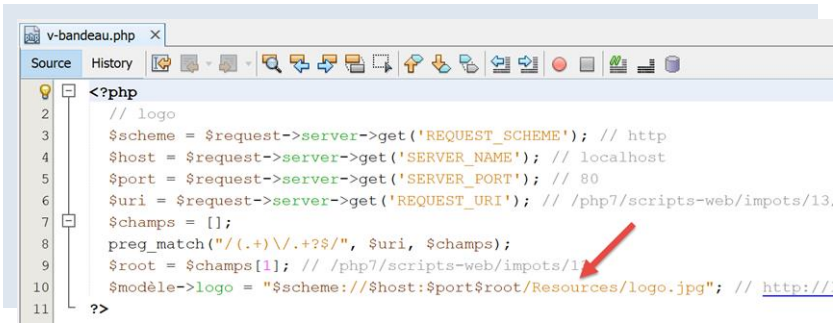
Dans le fichier **[Config/config.json]** :



Dans le fichier **[main.php]** :



Dans le fichier **[Views/v-bandeau.php]** :



2 Introduction au langage ECMASCRIPT6 par l'exemple

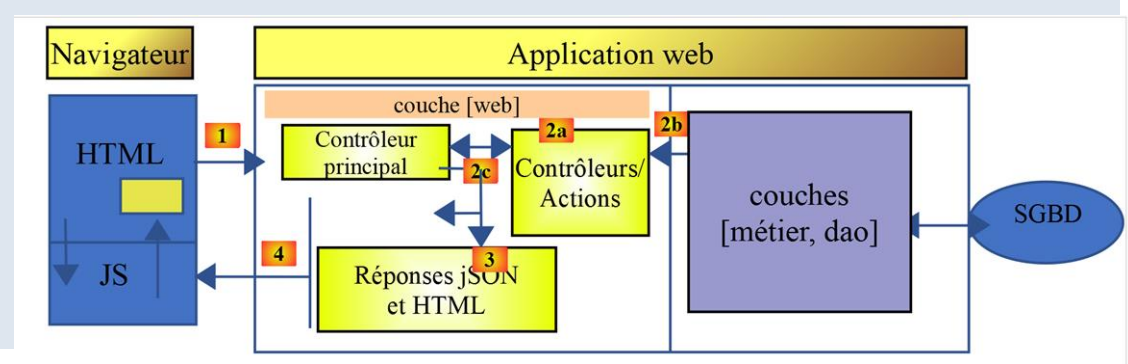
2.1 Présentation du cours

La |dernière version du serveur de calcul de l'impôt| développée dans le chapitre sur le langage PHP7 peut être améliorée de diverses manières :

- la version écrite est centrée sur le serveur. La tendance est désormais (juillet 2019) au client / serveur :
  - le serveur fonctionne en service jSON ;
  - une page statique ou non est le point d'entrée de l'application web. Cette page contient du HTML /CSS mais aussi du Javascript ;
  - les autres pages de l'application web sont obtenues dynamiquement par le Javascript :
    - la page HTML peut être obtenue par assemblage de fragments statiques, fournis par le même serveur qui a fourni la page d'accueil ou bien entièrement construite par le Javascript ;
    - ces différentes pages affichent des données qui sont demandées au service jSON ;

Ainsi le travail est réparti sur le client et le serveur. Le serveur ainsi déchargé peut servir davantage d'utilisateurs.

L'architecture correspondant à ce modèle est le suivant :



JS : Javascript

Le code javascript est client :

- d'un service de pages ou fragments statiques ou non ;
- d'un service jSON ;

Le code Javascript est donc un client jSON et à ce titre peut être organisé en couches **`[UI, métier, dao]`** (UI : User Interface) comme l'ont été nos clients jSON écrits en PHP. Au final, le navigateur ne charge qu'une unique page, la page d'accueil. Toutes les autres sont obtenues et construites par le Javascript. On appelle ce type d'application **SPA** : **S**ingle **P**age **A**pplication ou encore **APU** : **A**pplication à **P**age **U**nique.

Ce type d'application fait également partie des applications dites **AJAX** : **A**synchronous **J**avascript **A**nd **X**ML :

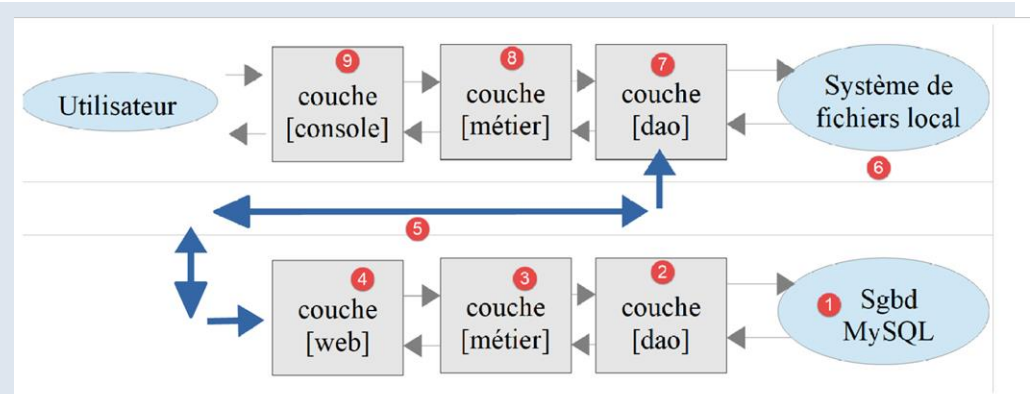
- **Asynchronous** : parce que les appels du client Javascript au serveur jSON sont asynchrones ;
- **XML** : parce que XML était la technologie utilisée avant l'avènement du jSON. On a cependant gardé l'acronyme AJAX ;

Nous allons étudier une telle architecture dans les chapitres à venir. Côté client, nous utiliserons le framework Javascript **`[Vue.js]`** **`[https://vuejs.org/]`** pour écrire le client Javascript du serveur jSON PHP que nous avons écrit dans le chapitre sur PHP7.

`[Vue.js]` est un framework Javascript. Pour le comprendre, il faut maîtriser ce langage. Nous présentons dans ce document la norme **ECMAScript 6** qui est la normalisation la plus récente (en 2019) de ce langage. On trouvera l'historique et le rôle d'ECMAScript sur Wikipedia **`[https://fr.wikipedia.org/wiki/ECMAScript]`**.

Cette section du cours propose une liste de **scripts console** Javascript dans différents domaines (structures du langage, accès aux bases de données, au réseau internet, programmation en couches, programmation par interfaces). La section se termine avec deux applications :

Une application console qui sera un client du serveur de calcul de l'impôt construit dans le document **[1]**. Ce client aura la même architecture que celle du client console PHP construit dans le chapitre sur PHP7 :



- les couches **[7-9]** seront celles du client Javascript s'exécutant dans une console ;
- les couches **[1-4]** sont celles du serveur PHP 7 construit dans le document **[1]** ;
- contrairement au client PHP console construit dans le document **[1]**, il n'y aura pas d'interactions avec le système de fichiers local **[6]** ;

On a là une application client / serveur où le client est une application console. Une seconde application sera écrite où le code des couches **[7-9] sera porté dans un navigateur**. Nous serons alors prêts à aborder les frameworks Javascript des navigateurs dans les documents **[3]** et **[4]**.

Les scripts de ce document sont commentés et leur exécution console reproduite. Des explications supplémentaires sont parfois fournies. Le document nécessite une lecture active : pour comprendre un script, il faut à la fois lire son code, ses commentaires et ses résultats d'exécution.

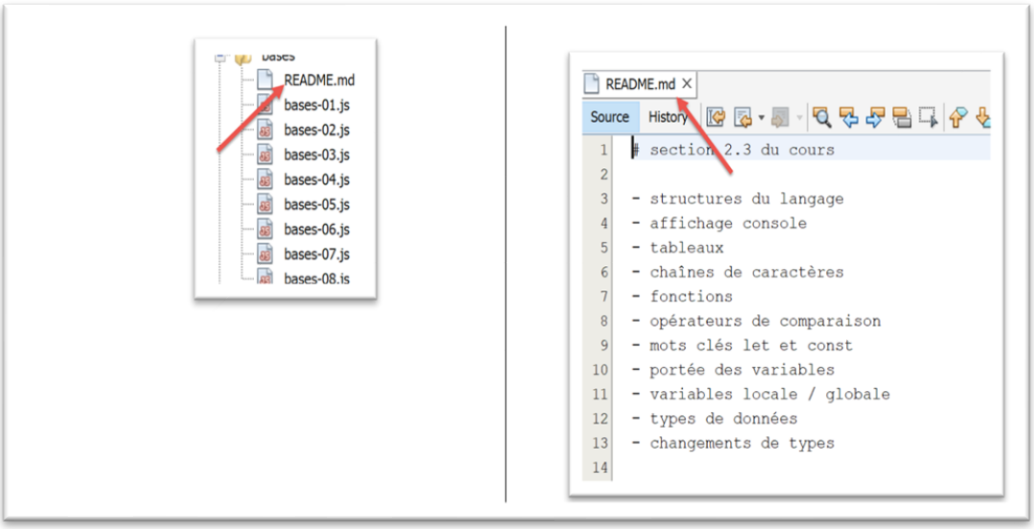
Les exemples du document sont disponibles |ici|.
L'application serveur PHP 7 peut être testée |ici|.

Serge Tahé, octobre 2019

Le contenu de cette partie est la suivante :

Chapitre 11	Programmation événementielle et fonctions asynchrones : émission / réception d'événements, synchronisation de fonctions asynchrones avec la classe **[Promise]**, synchronisation de fonctions asynchrones avec async / wait
Chapitre 12	Bibliothèques HTTP : bibliothèques axios et fetch, écriture d'un client jSON d'un serveur de calcul de l'impôt développé en PHP7
Chapitre 13	Les classes : une fonction peut être utilisée comme un objet, propriétés, méthodes, constructeur d'une classe, héritage
Chapitre 14	Clients HTTP du service de calcul de l'impôt : écriture d'un client console 1 jSON du serveur de calcul de l'impôt, écriture d'un client console 2 jSON du serveur de calcul de l'impôt, écriture d'un client 3 jSON du serveur de calcul de l'impôt, client exécuté par un navigateur

Certains lecteurs préfèreront lire, modifier et tester le code plutôt que de lire un cours. Dans chaque dossier des codes de ce document on a mis un fichier **[README.md]** résumant le contenu du dossier et faisant le lien avec le cours :



2.2 Installation d'un environnement de travail

Nous allons utiliser les outils suivants (sous Windows 10 x 64 bits) :

- **[Laragon]** pour exécuter le serveur web PHP ;
- **[Netbeans]** pour modifier le code PHP ;
- **[Visual Studio Code]** pour écrire les codes Javascript ;
- **[node.js]** pour les exécuter ;
- **[npm]** pour télécharger et installer les bibliothèques Javascript dont nous aurons besoin ;

2.2.1 Environnement de travail pour le serveur web

Les scripts PHP ont été écrits et testés dans l'environnement suivant :

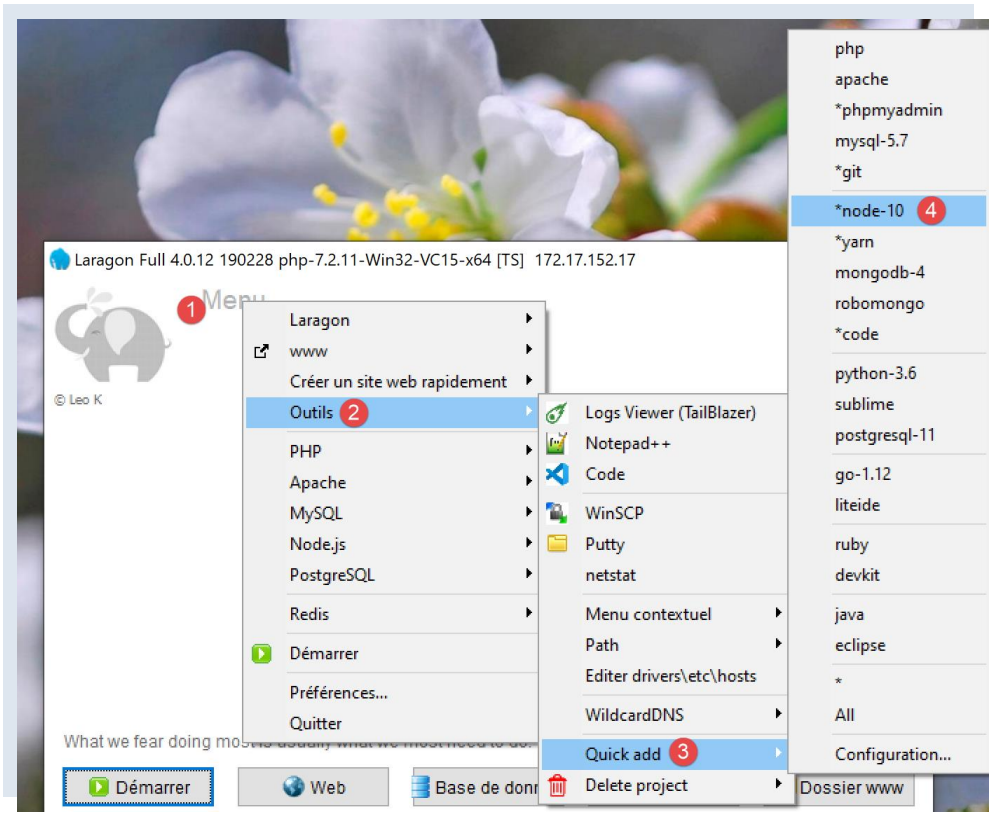
- un environnement serveur web Apache / SGBD MySQL / PHP 7.3 appelé **Laragon ;**
- l'IDE de développement **Netbeans 10.0 ;**

Ces deux outils ont déjà été installés dans l'étude de PHP 7 :

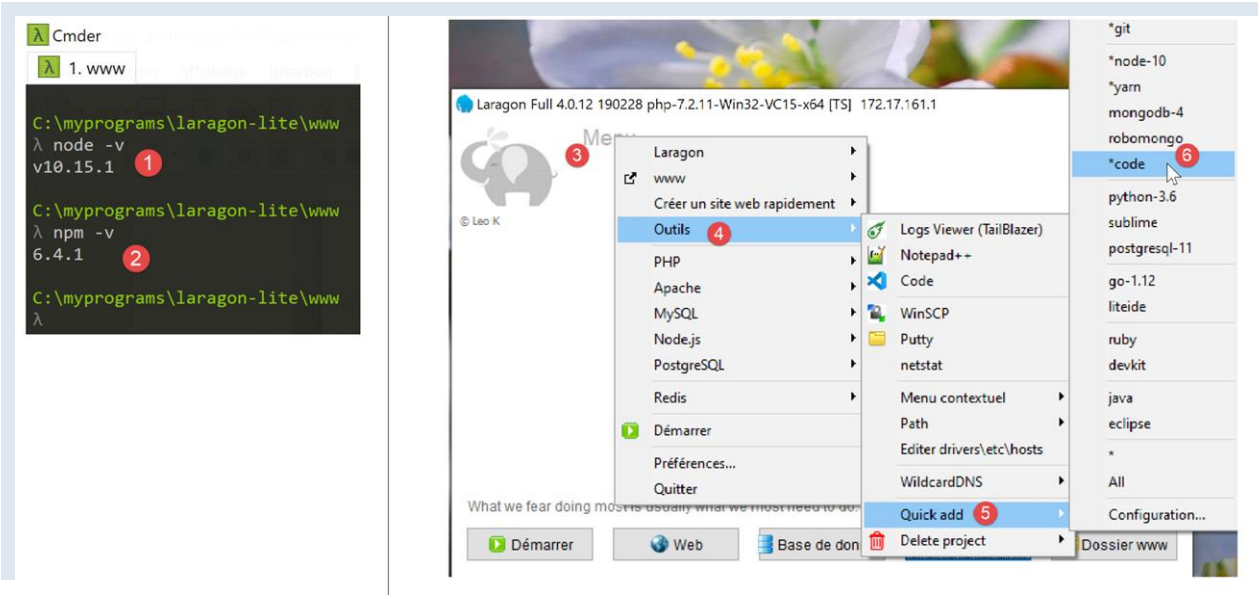
- l'installation de Laragon est décrite |ici| ;
- l'installation de Netbeans est décrite |ici] ;

2.2.2 Environnement de travail pour JavaScript

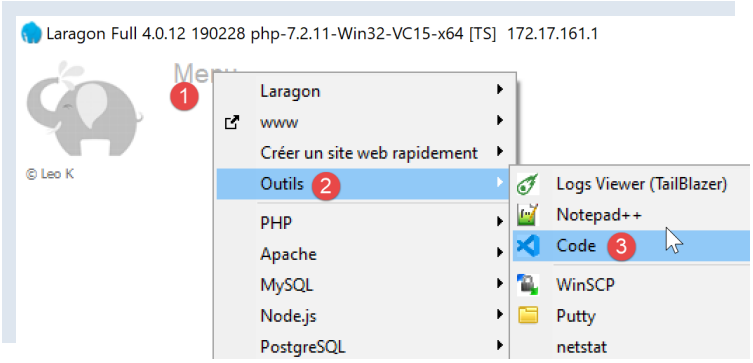
Ces outils peuvent être installés à partir de Laragon (cf. paragraphe lien) :

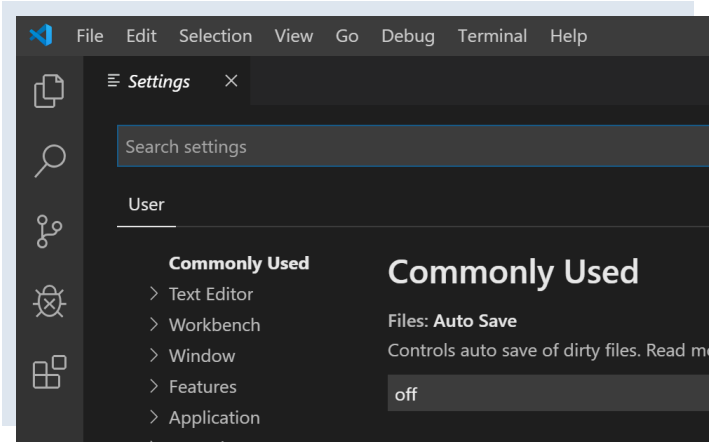


En **[4]**, on installe **[node.js]**. Une fois l'installation terminée, on ouvre un terminal Laragon (cf. paragraphe lien) et on demande la version de **[node.js]** installée (1) ainsi que celle de **[npm]** (2) :



Ensuite nous installons Visual Studio Code appelé fréquemment **[code]** ou **[VSCode]** **[3-6]**. Ceci fait, nous pouvons lancer cet outil de développement :

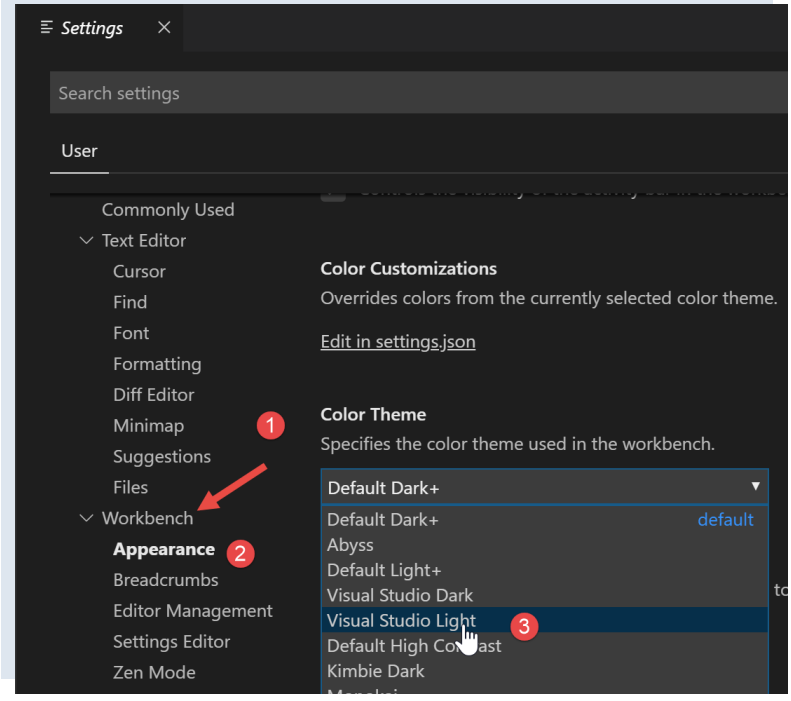




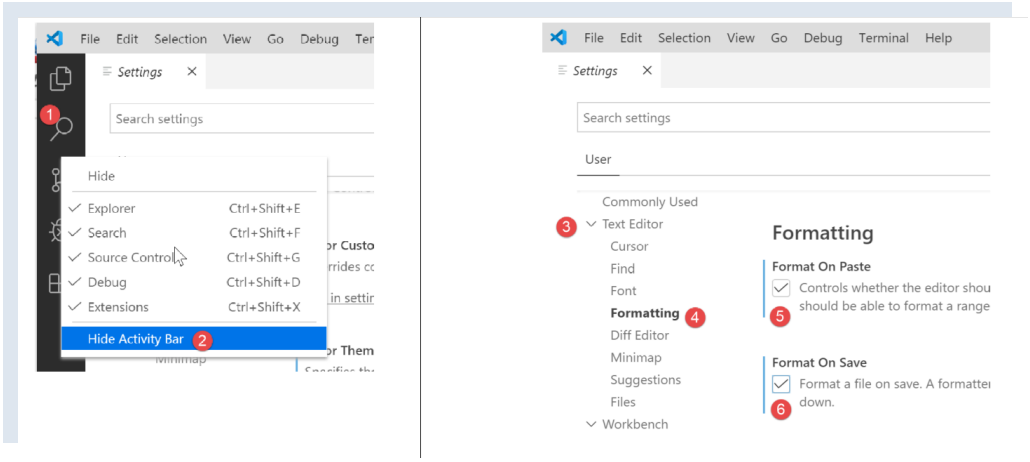
2.2.2.1 Configuration de Visual Studio Code

Nous montrons maintenant comment nous avons configuré **[VSCode]** afin que le lecteur comprenne les copie d'écran qui apparaîtront de temps en temps. Le lecteur est lui libre de configurer **[VSCode]** comme il l'entend. Il peut même installer son environnement de travail favori. Celui-ci importe peu pour ce que nous allons faire par la suite.

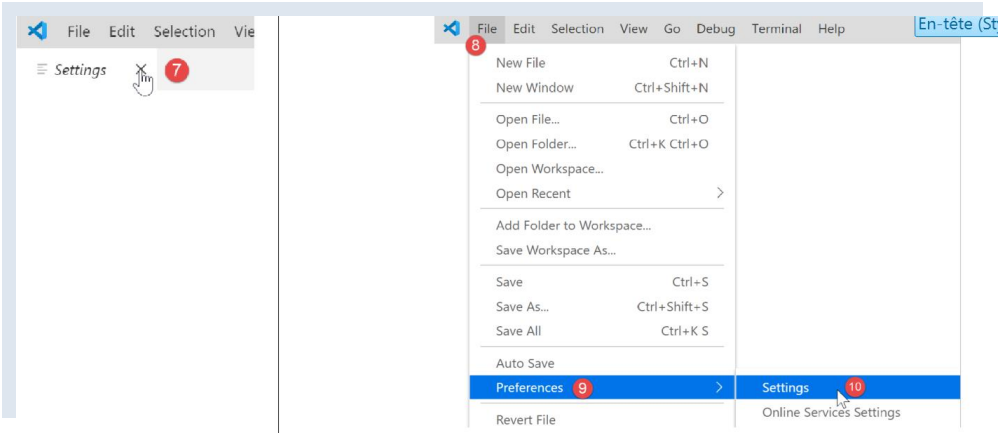
Tout d'abord, nous changeons l'apparence de la fenêtre **[VSCode]** pour avoir un fond clair plutôt que noir :



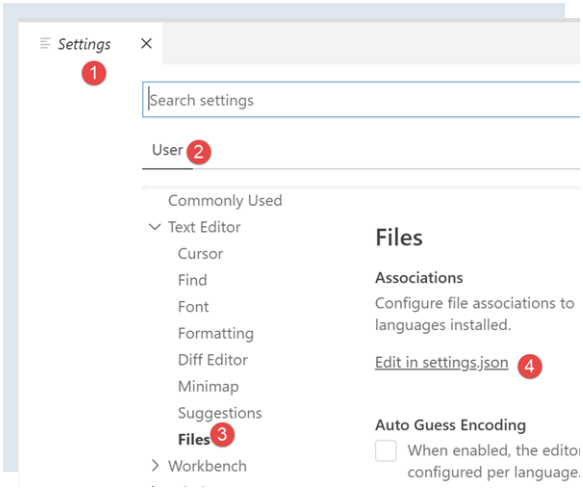
Puis nous cachons la barre de gauche **[1-2]** dont les éléments sont également disponibles dans le menu. En **[3-6]**, nous demandons un formatage du code à chaque sauvegarde du fichier et à chaque copier / coller.



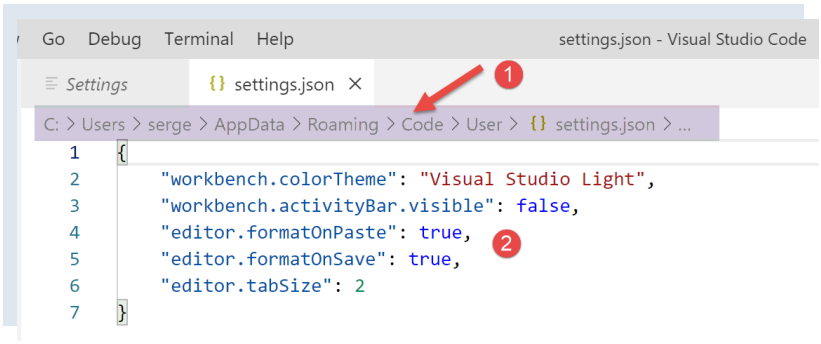
Après avoir sauvegardé la configuration **[Ctrl-S]**, on peut fermer la fenêtre **[Settings] [7]**. On peut revenir à tout moment à la configuration de **[VSCode] [8-10]** :



Ces configurations sont sauvegardées dans un fichier **[settings.json]** que l'on peut éditer directement. Ouvrons la fenêtre de configuration **[Settings]** comme il a été vu :

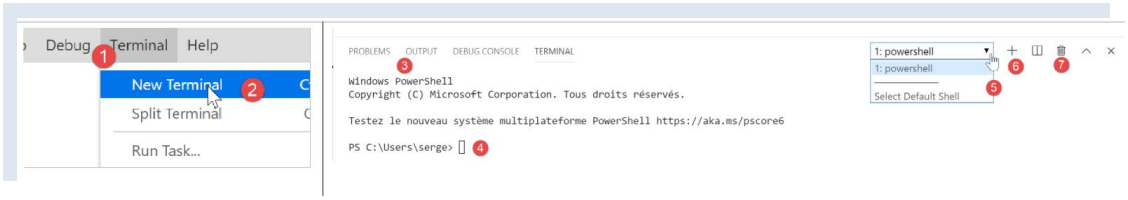


En **[4]**, on peut éditer directement le fichier **[settings.json]** :



- en **[1]**, le chemin du fichier **[settings.json]**. Une façon de revenir à la configuration par défaut est de supprimer ce fichier ;
- en **[2]**, les configurations que nous venons de faire ;

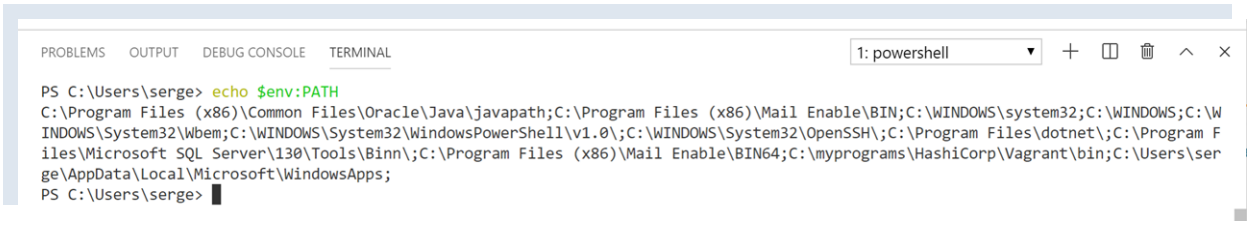
Maintenant, ouvrons un terminal à l'intérieur de **[VSCode] [1-2]** :



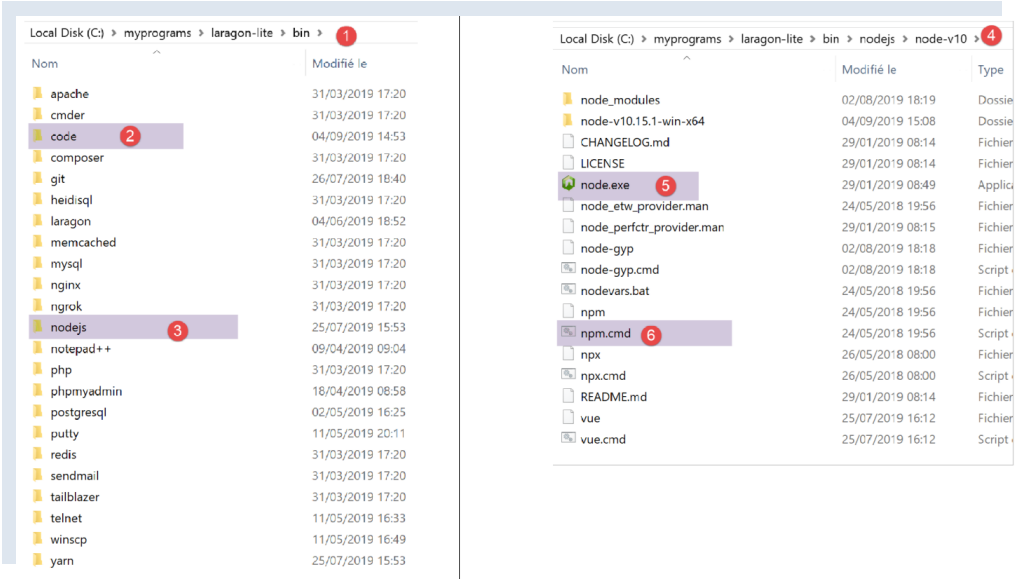
- en **[3]**, le type de terminal ouvert, ici PowerShell ;
- en **[4]**, on peut taper des commandes Windows ;
- en **[6]**, on peut ouvrir d'autres terminaux ;
- en **[5]**, la liste des terminaux ouverts ;
- en **[7]**, supprime le terminal actif ;

Nous utiliserons le terminal de **[VSCode]** pour installer des packages (bibliothèques) Javascript avec l'outil **[npm]** (Node Package Manager). Demandons, comme nous l'avons fait précédemment dans un terminal Laragon, la version de **[npm]** installée :

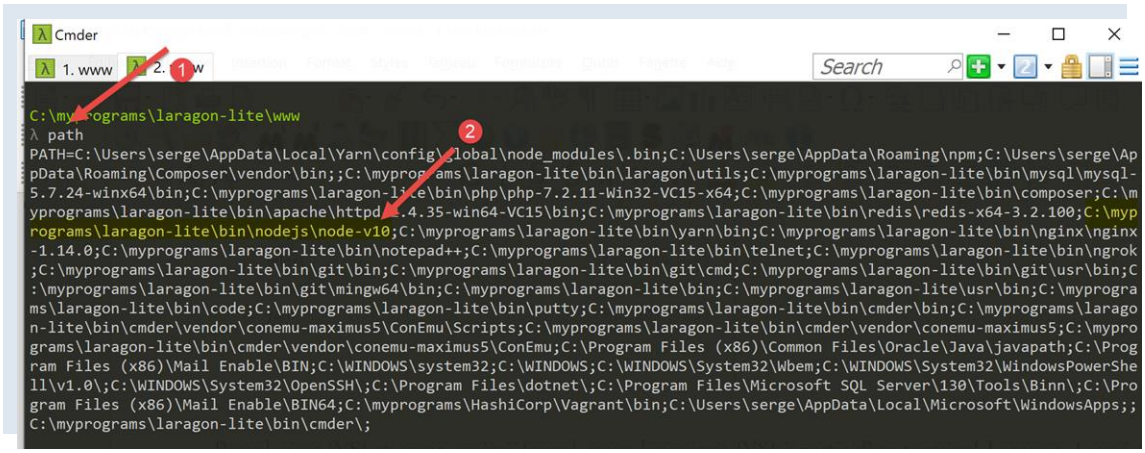
On voit que la commande **[npm]** n'a pas été reconnue. Cela signifie qu'elle n'appartient pas au PATH (liste des dossiers à explorer pour chercher un exécutable, ici **[npm]**) du terminal. On peut connaître le PATH utilisé par le terminal :



L'exécutable **[npm]** ne se trouve pas parmi ces dossiers. Comme les autres outils installés par Laragon, il se trouve dans le dossier **[<laragon>\bin]** de Laragon et plus précisément dans le dossier de **[nodejs] [4-6]**.

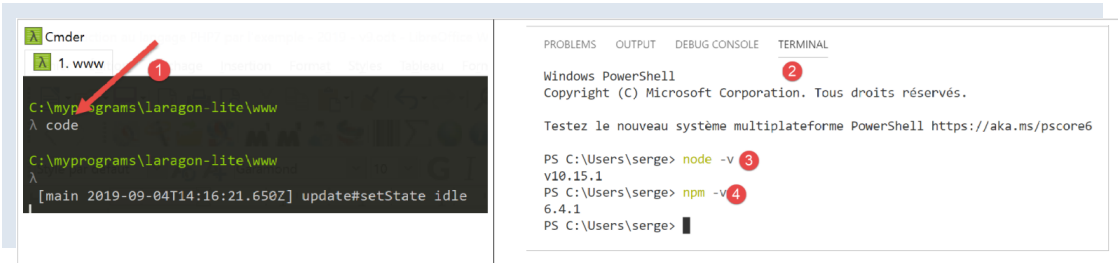


Pour lancer **[VSCode]** et avoir accès à **[npm]**, nous lancerons **[VSCode]** à partir d'un terminal Laragon. Lancé de cette façon, **[VSCode]** va hériter du PATH du terminal Laragon qui lui, contient le dossier des exécutables **[node]** et **[npm]** :



- en **[1]** : on tape la commande **[path]** ;
- en **[2]** : la liste des dossiers du PATH. On y voit le dossier **[node-v10]** **[2]**, ce qui nous garantit que les exécutables **[node]** et **[npm]** seront trouvés ;

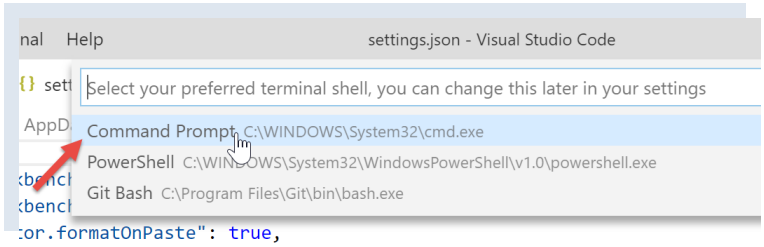
[VSCode] est lancé à partir d'un terminal Laragon avec la commande **[code]** :



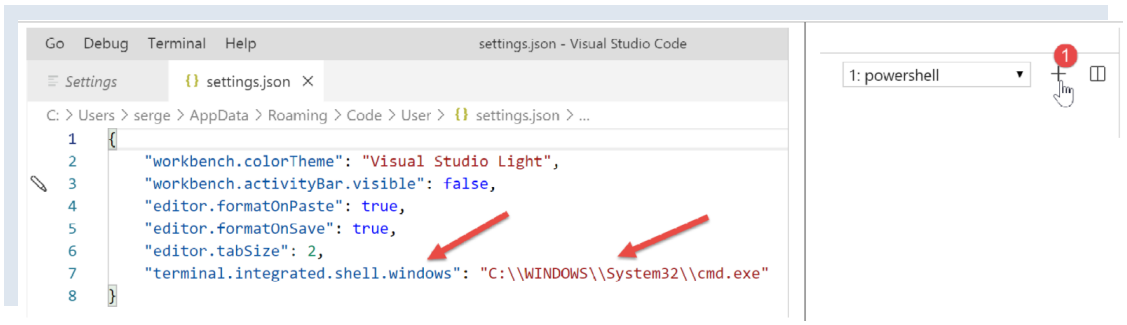
- en **[2]**, on ouvre un terminal PowerShell dans **[VSCode]** ;
- en **[3-4]**, on voit que les exécutables **[node]** et **[npm]** sont accessibles ;

Note : il ne faut pas fermer le terminal Laragon qui a lancé l'environnement de développement **[VSCode]**, sinon VSCode lui-même se ferme.

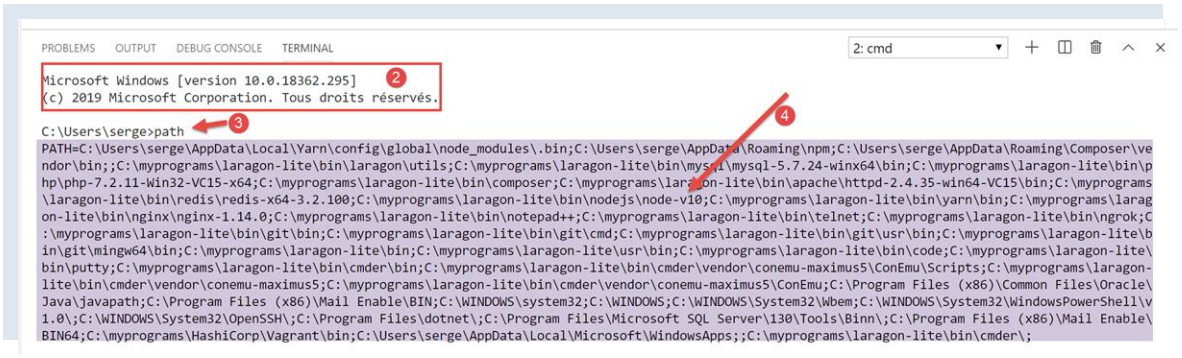
Nous allons faire une dernière configuration : nous allons changer le terminal par défaut de **[VSCode]** :



Le fichier **[settings.json]** se met aussitôt à jour :



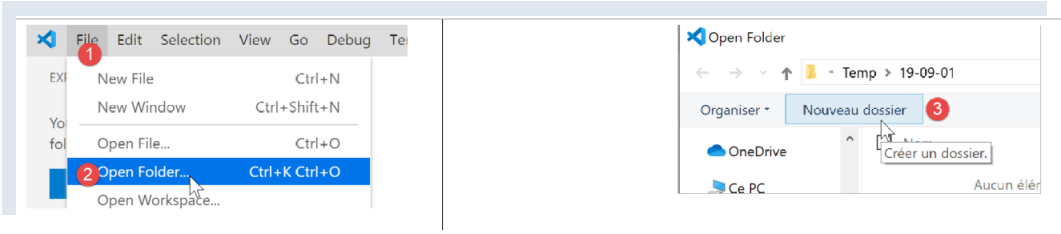
Maintenant, ouvrons un nouveau terminal **[VSCode] [1]** :

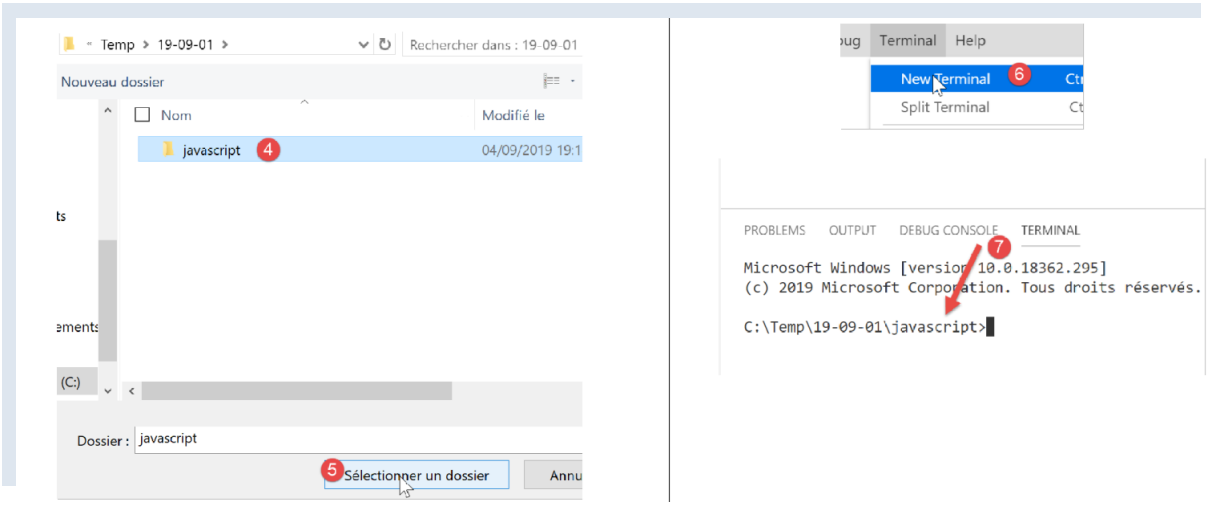


- en **[2]**, un terminal **[cmd]** (pas PowerShell) ;
- en **[3]**, la commande **[path]** donne le PATH du terminal ;
- en **[4]**, on y voit bien le dossier des exécutables **[node]** et **[npm]**

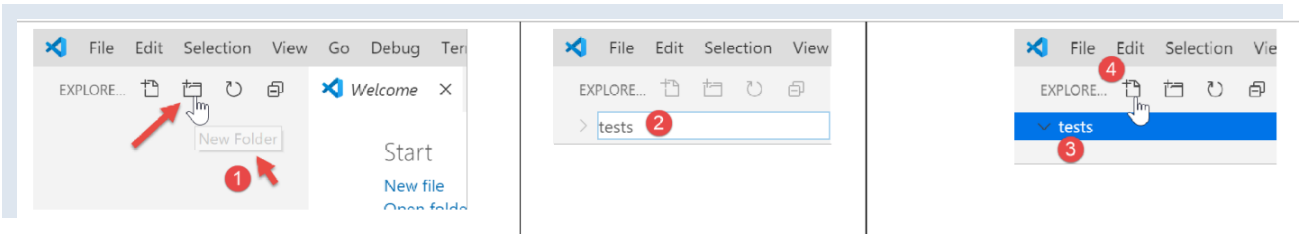
2.2.2.2 Ajout d'extensions à Visual Studio Code

Créons un fichier Javascript avec **[VSCode]** :

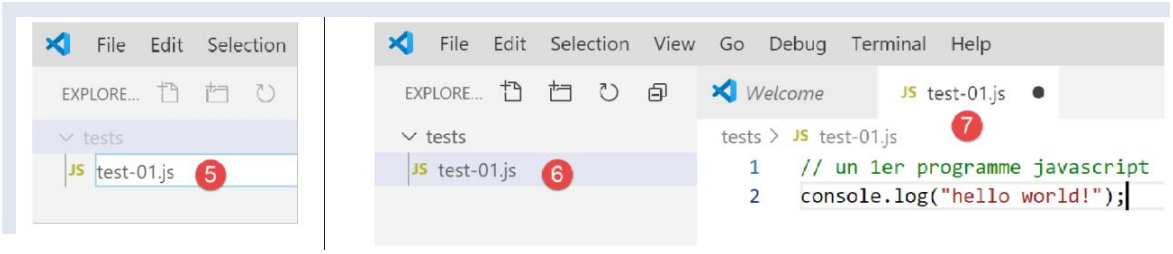




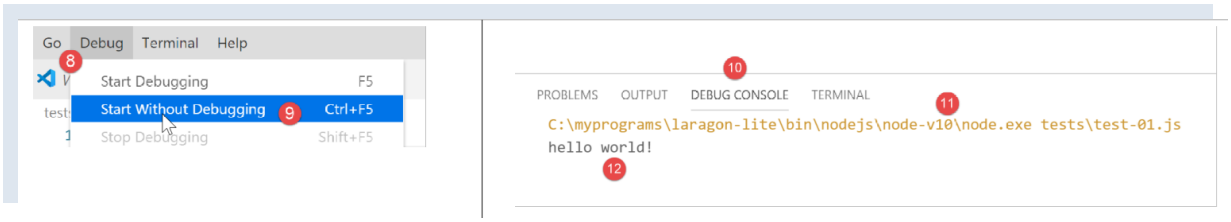
- en **[3-4]**, on crée un dossier ;
- en **[5]**, on en fait le dossier courant de **[VSCode]** ;
- en **[6]**, on ouvre un terminal ;
- en **[7]**, on voit qu'on est positionné sur le dossier choisi. Les opérations à suivre vont se faire dans celui-ci ;



- en **[1-3]** : on crée un nouveau dossier ;
- en **[4]** : on ajoute un fichier dans ce dossier ;

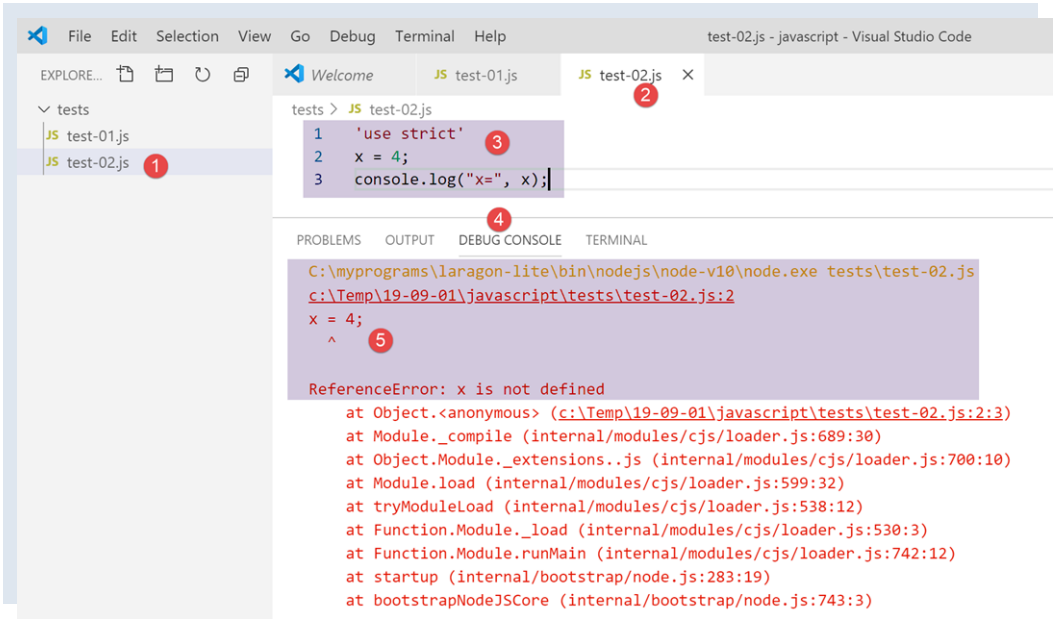


- en **[5-7]** : on crée notre 1er programme Javascript ;



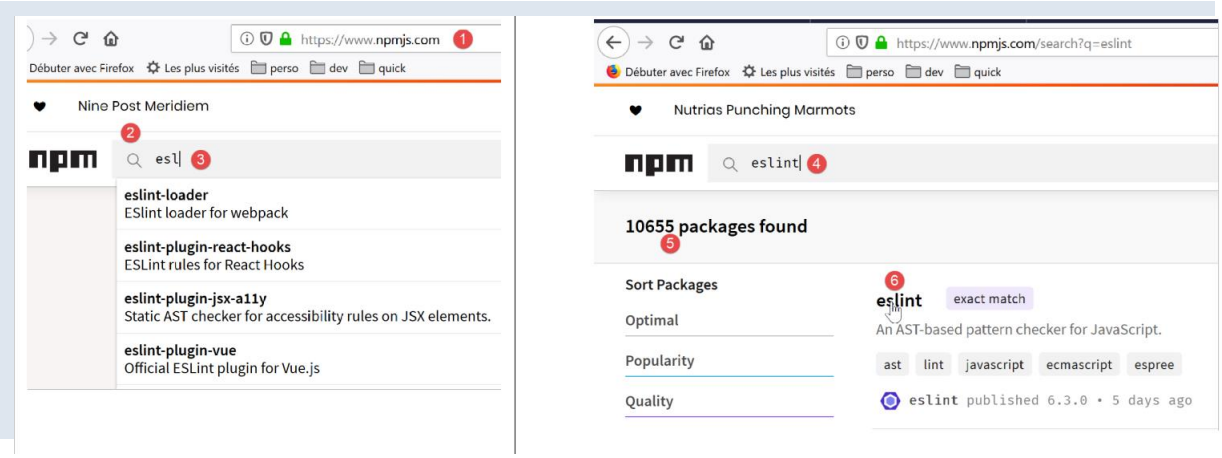
- en **[8-9]** : on exécute le programme Javascript ;
- le résultat apparaît dans la console d'exécution **[10]**. On voit en **[11]** la commande qui a été exécutée : c'est l'application **[node]** qui a exécuté le script **[test-01.js]**. C'est parce que cet exécutable est dans le PATH de **[VSCode]** que cela a pu être possible, sinon on aurait eu une erreur indiquant que la commande **[node]** n'était pas connue ;

Procédons de la même façon pour exécuter un second script **[test-02.js]** :

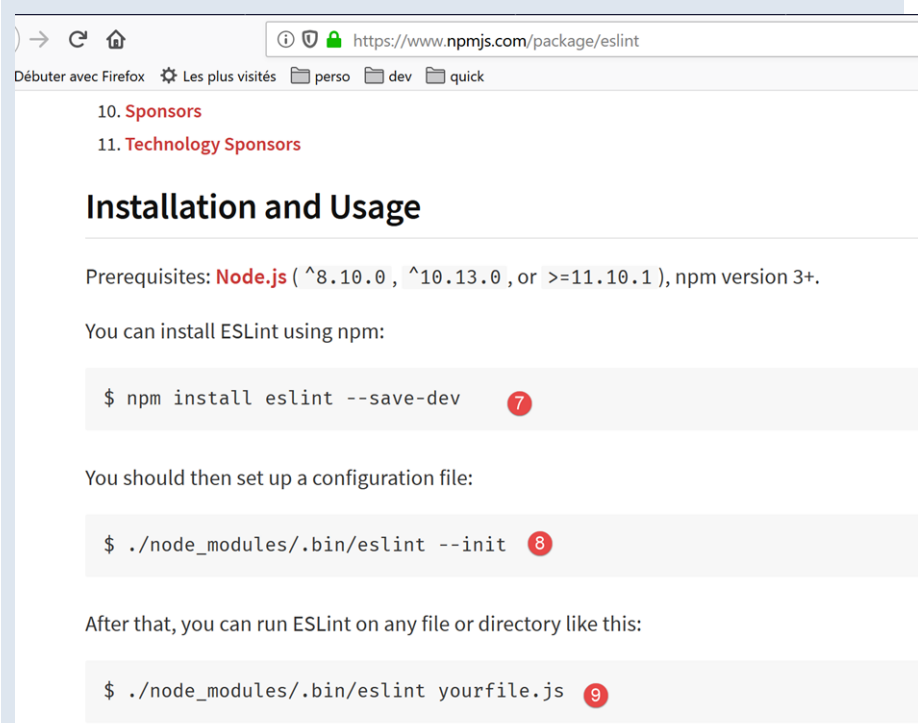


- en **[1-3]**, on définit le nouveau script. L'instruction **[use strict]** de la ligne 1 demande une vérification stricte de la syntaxe. Dans ce contexte, toute variable doit être déclarée avec l'un des mots clés **[let, const, var]**. Ce n'est pas le cas de la variable **[x]** de la ligne 2 ;
- lorsqu'on exécute ce code par **[Ctrl-F5]**, on obtient l'erreur **[5]**. Il est possible d'être averti de ce type d'erreur avant l'exécution. C'est préférable. Nous allons faire deux choses :
  - installer avec **[npm]** une bibliothèque appelée **[eslint]** qui vérifie que la syntaxe du script est conforme à la norme ECMAScript 6 ;
  - installer une extension à Visual Studio Code, appelée elle-aussi ESLINT qui facilite l'utilisation de la bibliothèque **[eslint]** au sein de **[VSCode]** ;

Installons tout d'abord la bibliothèque Javascript **[eslint]** à l'aide de l'outil **[npm]**. Pour installer une bibliothèque (on dit un package) **[npm]**, il faut connaître son nom exact. Si ce n'est pas le cas, on peut aller sur le site de **[npm]** à l'URL (2019) **[https://www.npmjs.com/]** :

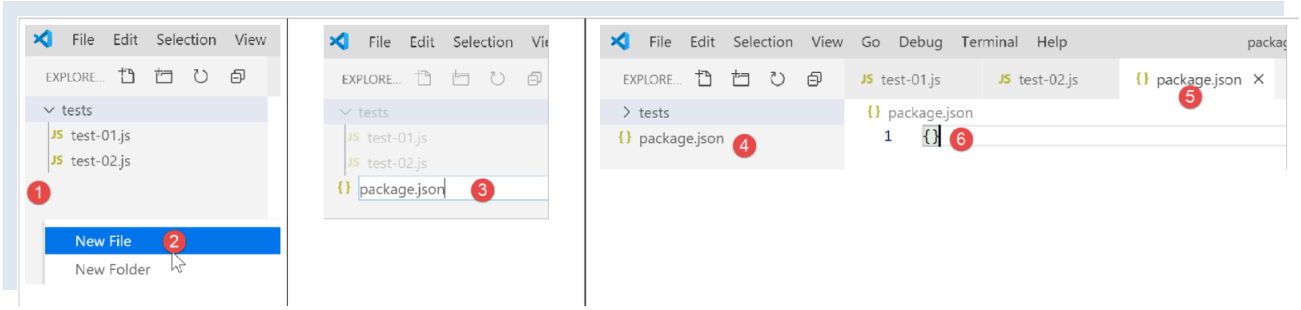


- en **[3]**, les packages commençant par **[esl]** ;
- en **[4-6]**, on trouve le package **[eslint]** ;



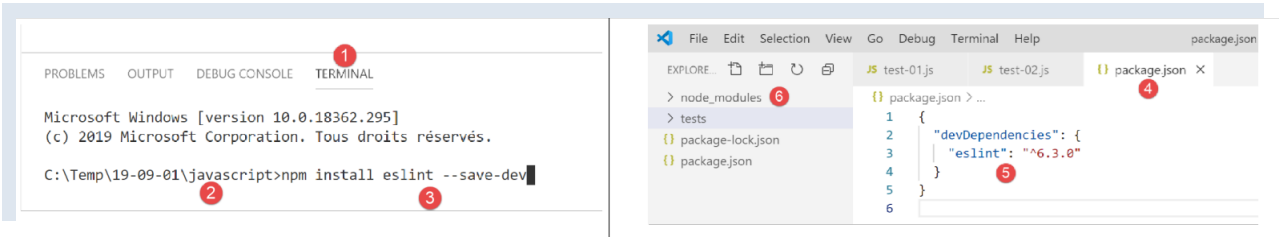
- en **[7]**, la commande **[npm]** pour installer le package **[eslint]** ;
- en **[8]**, la configuration du package ;
- en **[9]**, son utilisation pour vérifier la syntaxe d'un script Javascript ;

Nous installons le package **[eslint]** dans une fenêtre **[Terminal]** de **[VSCode]**. Tout d'abord, il nous faut créer un fichier **[package.json]** à la racine du dossier de travail de **[VSCode]**. Ce fichier contiendra la liste des packages jS utilisés par le projet **[VSCode]** :

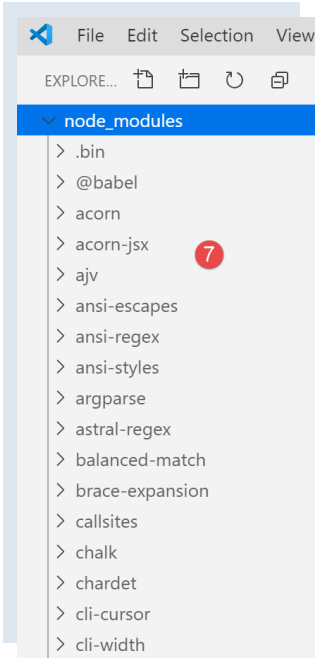


- en **[1]**, cliquer droit dans l'explorateur de projet (pas sur le dossier tests) ;
- en **[3-4]**, on crée le fichier **[package.json]** à la racine du projet **[javascript]**, au même niveau que le dossier **[tests]** (mais pas dans **[tests]**) ;
- en **[4-6]**, on met dans le fichier **[package.json]** un objet jSON vide ;

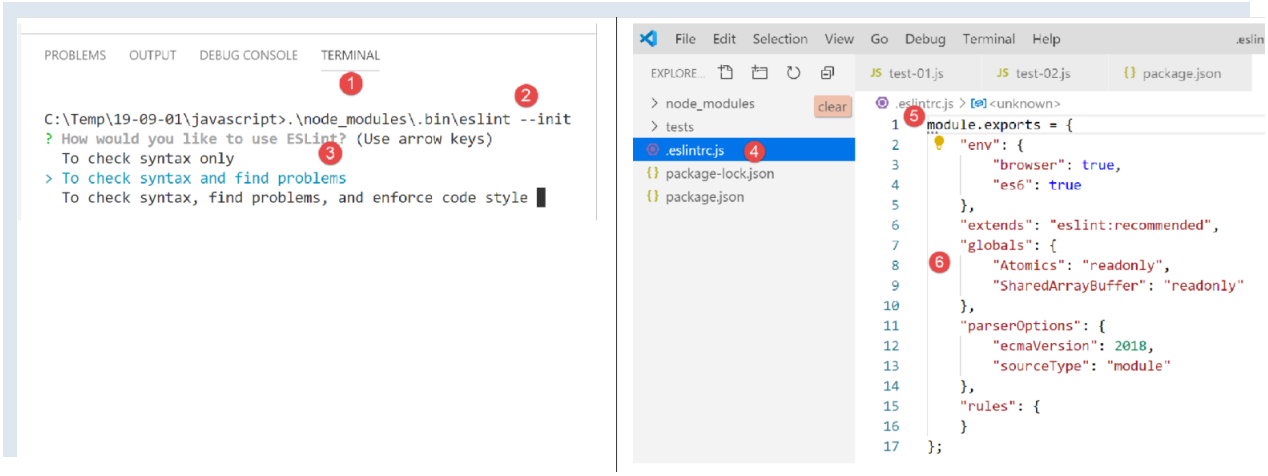
Puis on ouvre un terminal **[VSCode]** pour installer **[eslint]** :



- en **[2]**, on est à la racine du projet **[javascript]** ;
- en **[3]**, la commande qui installe le package **[eslint]** ;
- après exécution,
  - en **[4-5]**, le fichier **[package.json]** a été modifié. Ligne 3, on trouve la version de **[eslint]** installée. Ligne 2, **[devDependencies]** correspond à l'argument **[--save-dev]** de l'installation. Cet argument signifie que la dépendance installée doit être inscrite dans le fichier **[package.json]** comme élément de la propriété **[devDependencies]**. Cette propriété liste les dépendances du projet dont on a besoin en mode développement mais pas en mode production. En effet, on a besoin de la dépendance **[eslint]** uniquement en développement pour vérifier que le code écrit respecte la norme ECMAScript ;
  - en **[6]**, un dossier **[node_modules]** est apparu dans le projet. C'est le dossier où sont installées les dépendances du projet ;



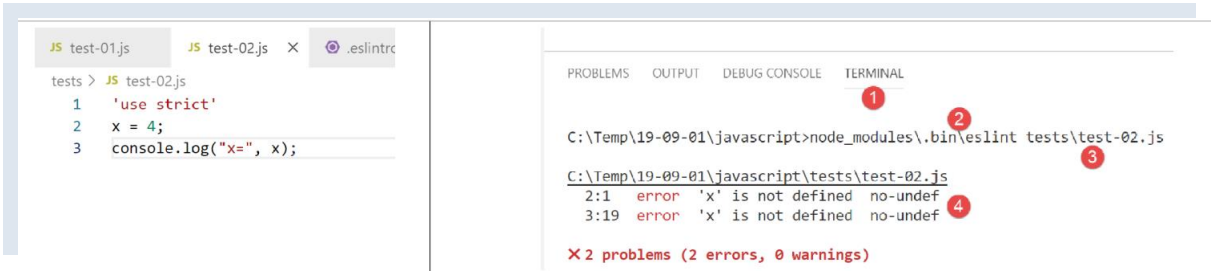
- en **[7]**, une partie des packages installés. Ceux-ci sont très nombreux. En effet, non seulement le package **[eslint]** a été installé mais également tous les packages sur lesquels celui-ci s'appuie ;



- **[1-2]**, dans un terminal **[VSCode]** on émet la commande de configuration du package **[eslint]**. Celle-ci va poser diverses questions **[3]** pour savoir comment on souhaite utiliser **[eslint]**. Dans le doute, laissez les options proposées par défaut. Pour sélectionner une option, utilisez les flèches haute et basse du clavier pour choisir l'option puis validez celle-ci ;
- en **[4]**, un fichier **[.eslintrc.js]** a été créé à la racine du projet ;
- en **[6]**, le contenu du fichier. Vous pouvez en copier le contenu dans votre propre fichier ;

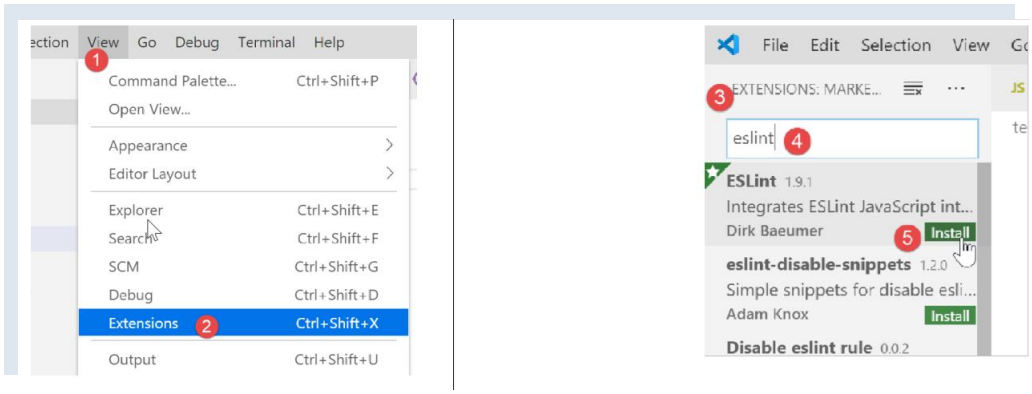
```
1.  module.exports = {
2.    "env": {
3.      "browser": true,
4.      "es6": true
5.    },
6.    "extends": "eslint:recommended",
7.    "globals": {
8.      "Atomics": "readonly",
9.      "SharedArrayBuffer": "readonly"
10.   },
11.   "parserOptions": {
12.     "ecmaVersion": 2018,
13.     "sourceType": "module"
14.   },
15.   "rules": {
16.   }
17. };
```

Tout ceci n'est pas suffisant pour signaler les erreurs du fichier **[test-02.js]** :

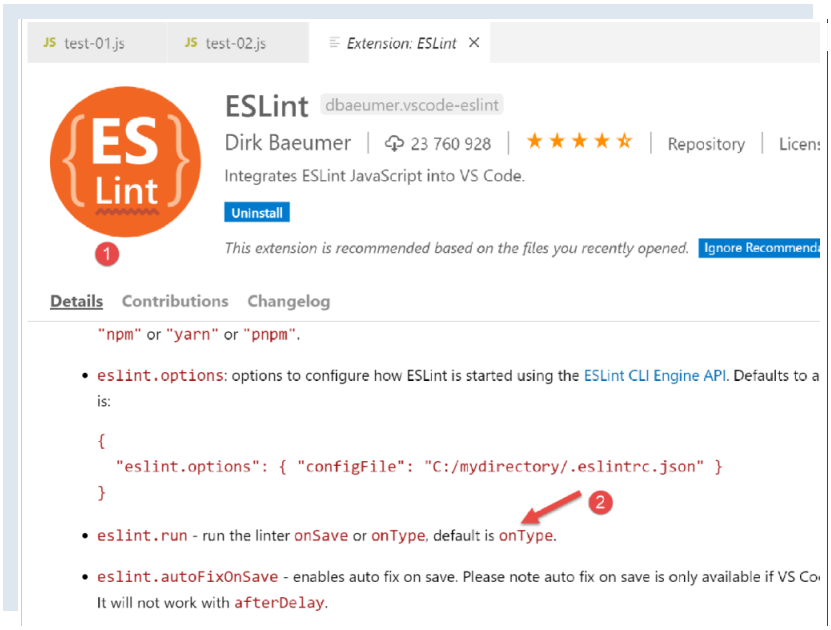


- il faut taper la commande **[2-3]** pour que le fichier **[tests/test-02.js]** soit analysé ;
- en **[4]**, l'erreur sur la variable non déclarée est détectée ;

Nous allons ajouter à **[VSCode]** une extension qui va permettre de voir les erreurs Javascript en temps réel. Cette extension s'appuie sur le package **[eslint]** que nous avons installé :

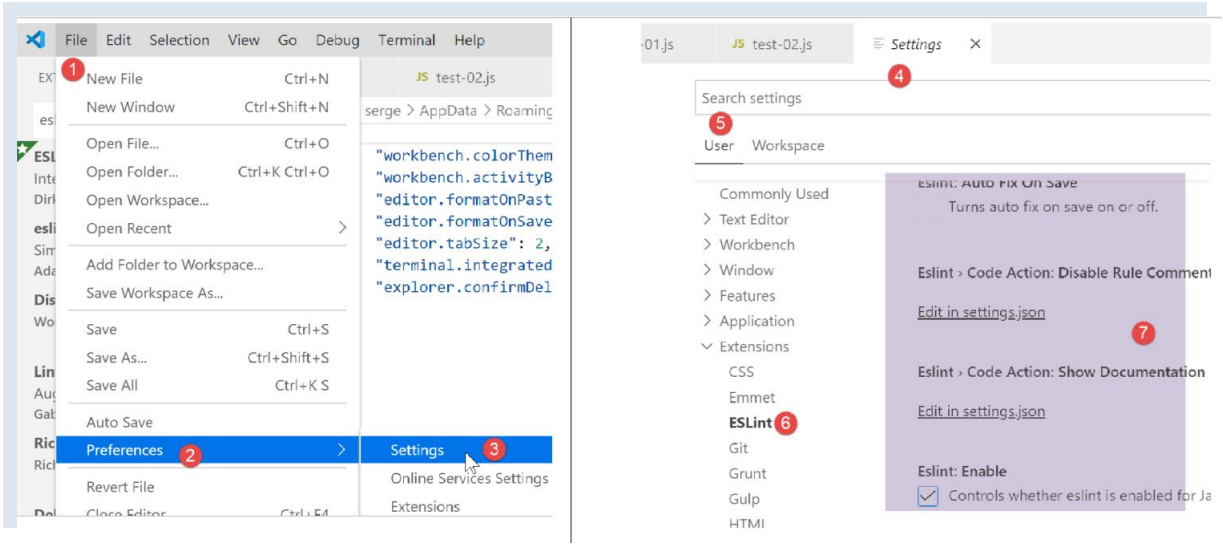


- en **[3-5]**, nous installons l'extension appelée **[ESLint]** ;



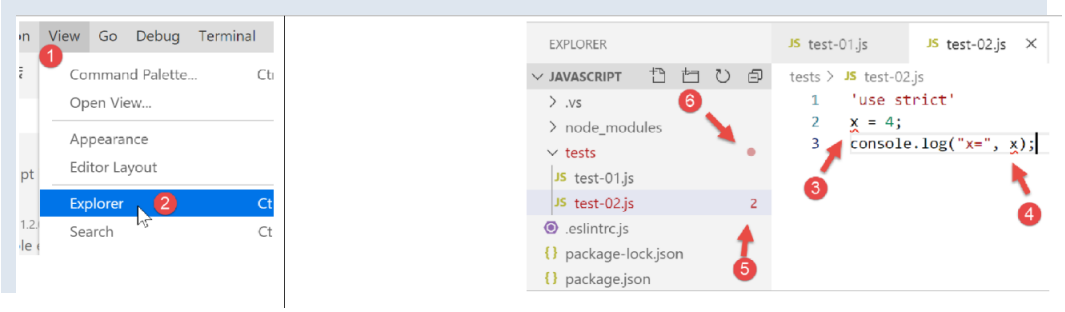
- en **[1]**, une page d'informations sur l'extension nouvellement installée ;
- en **[2]**, on voit que le mode de vérification de **[ESLint]** est **[type]**, ce qui signifie que la syntaxe des scripts jS sera vérifiée en même temps que la frappe du texte ;

ESLint peut être configuré via le fichier de configuration général de **[VSCode]** :



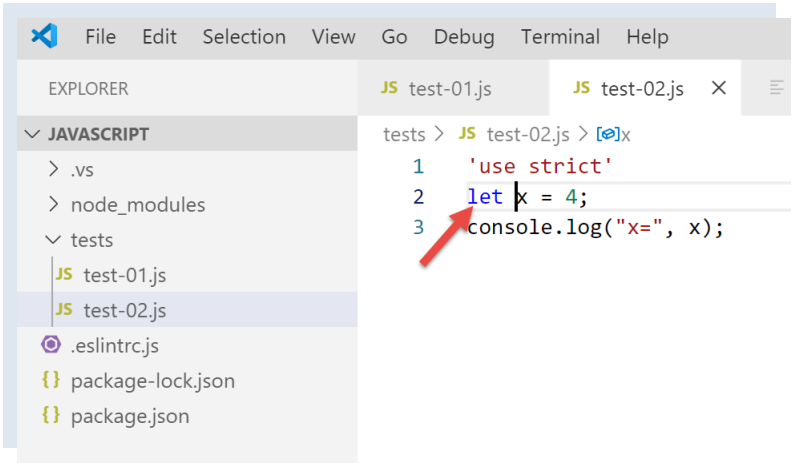
- en **[6-7]**, la configuration de **[ESLint]**. C'est ici que vous pourrez la modifier ;

Revenons maintenant au fichier **[test-02.js]** :

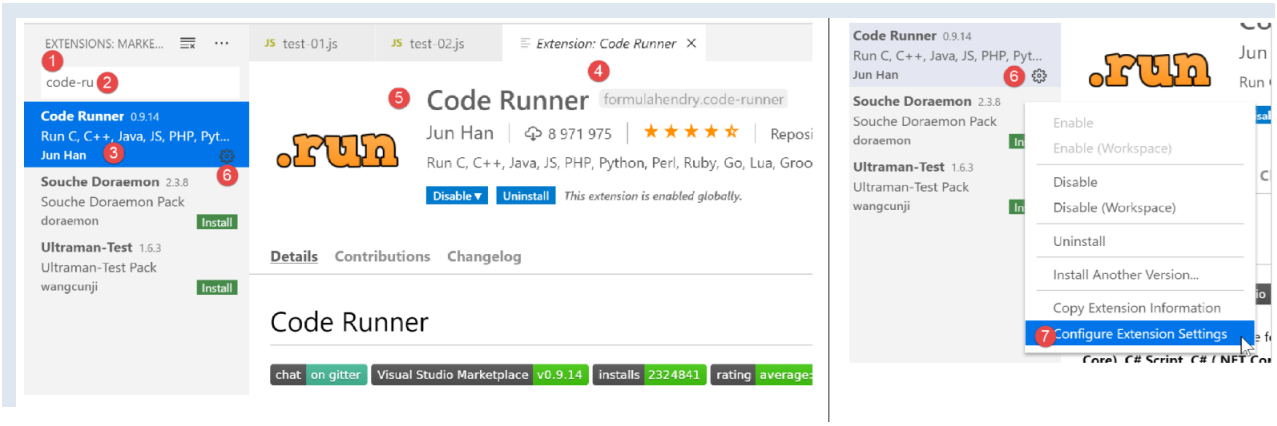


- en **[3-4]**, les erreurs sur la variable **[x]** sont désormais signalées ;
- en **[5]** : le nombre d'erreurs ESLint dans le fichier ;
- en **[6]**, indique que dans le dossier **[tests]**, il y a des fichiers erronés ;

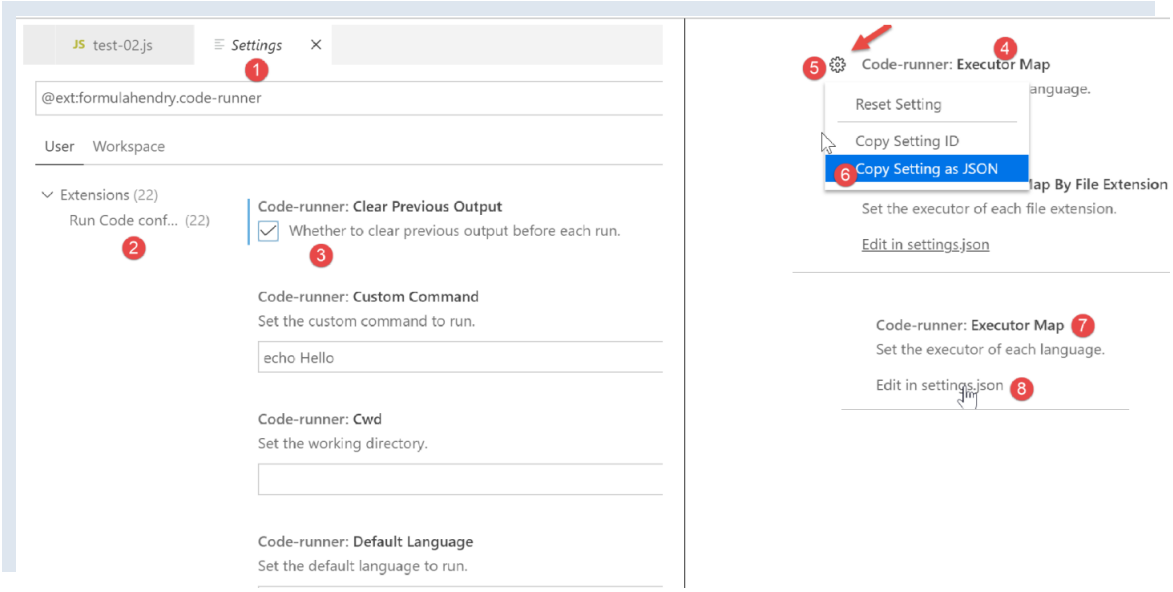
Si on corrige l'erreur, les avertissements d'ESLint disparaissent :



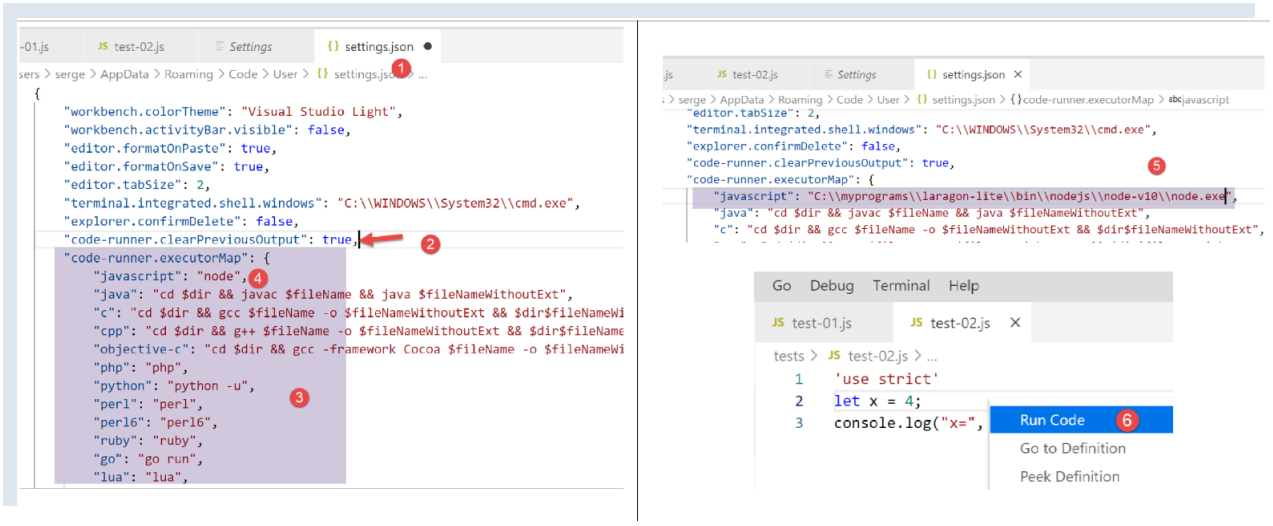
Installons maintenant une extension appelée **[Code Runner]** :



- une fois installée l'extension **[Code-Runner] [1-5]**, on peut la configurer avec **[6-7]** (ci-dessus) ;

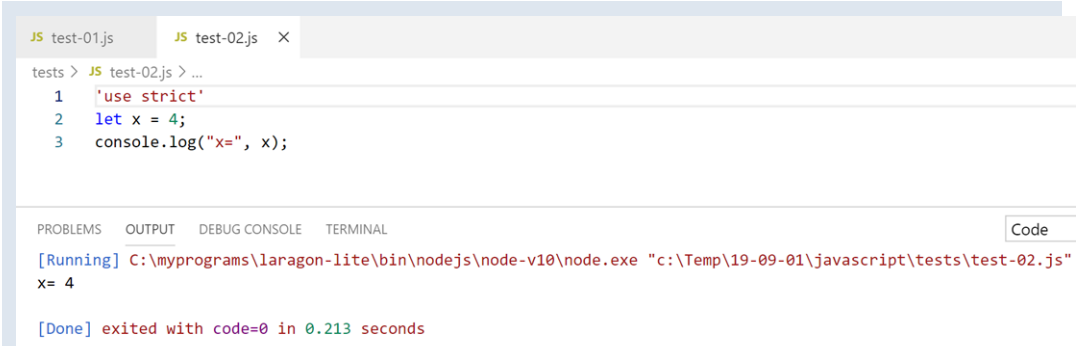


- en **[1-2]**, les éléments de configuration de **[Code-Runner]** ;
- en **[3]**, on demande à ce que le terminal de sortie soit nettoyé avant chaque exécution ;
- en **[4]**, on localise l'élément **[Executor Map]** qui liste les outils d'exécution de différents langages ;
- en **[5-6]**, on copie la configuration dans le presse-papiers ;
- en **[7-8]**, on modifie le fichier **[settings.json]** ;



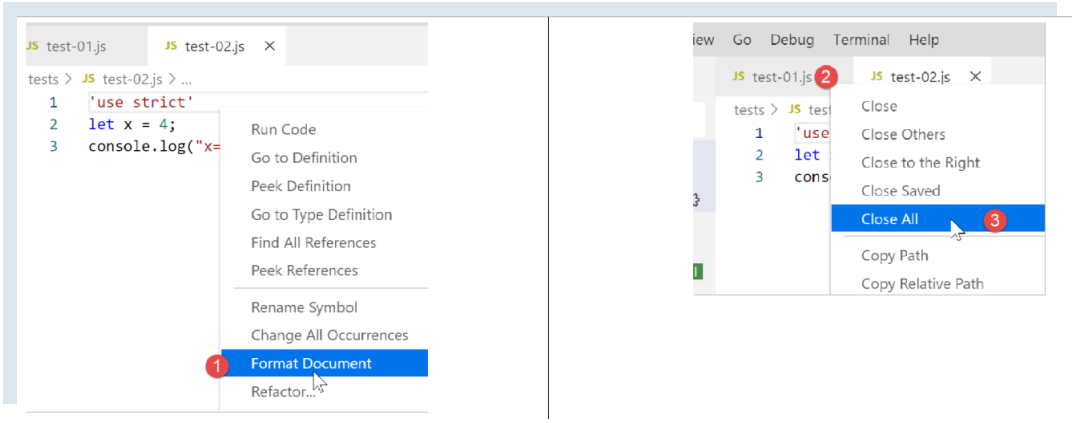
- en **[2]**, on ajoute la virgule derrière le dernier élément du fichier **[settings.json]** **[1]** ;
- en **[3]**, on colle ce qu'on a copié en **[5-6]** précédemment : c'est la liste des commandes permettant d'exécuter les différents langages supportés par **[VSCode]** ;
- en **[4]**, la commande permettant d'exécuter les fichiers Javascript. Celle-ci ne fonctionne que si **[node]** est dans le PATH de **[VSCode]**. Si ce n'est pas le cas, on peut mettre le chemin complet de l'exécutable **[5]** ;

Ceci fait sauvegardons la configuration (Ctrl-S). Avec l'extension **[Code Runner]**, les fichiers Javascript peuvent être exécutés avec un clic droit sur le code **[6]** (ci-dessus) :

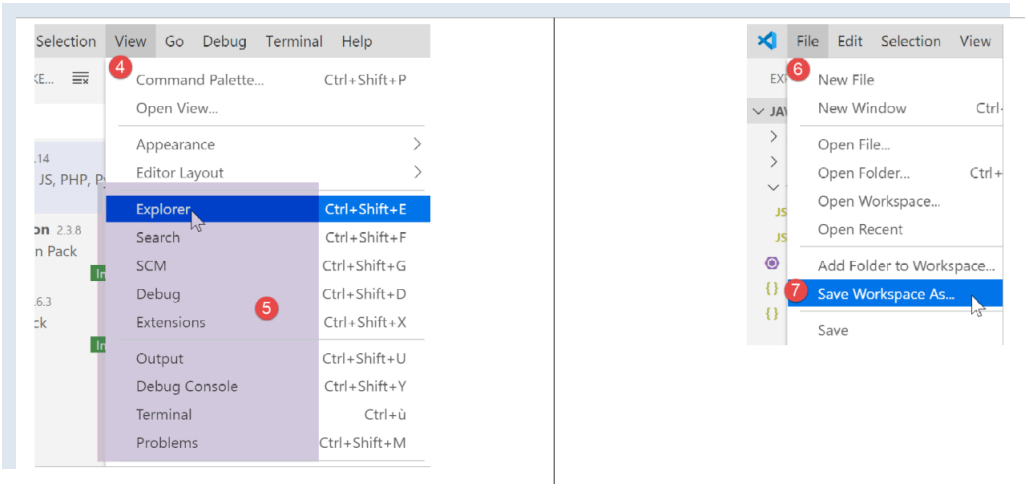


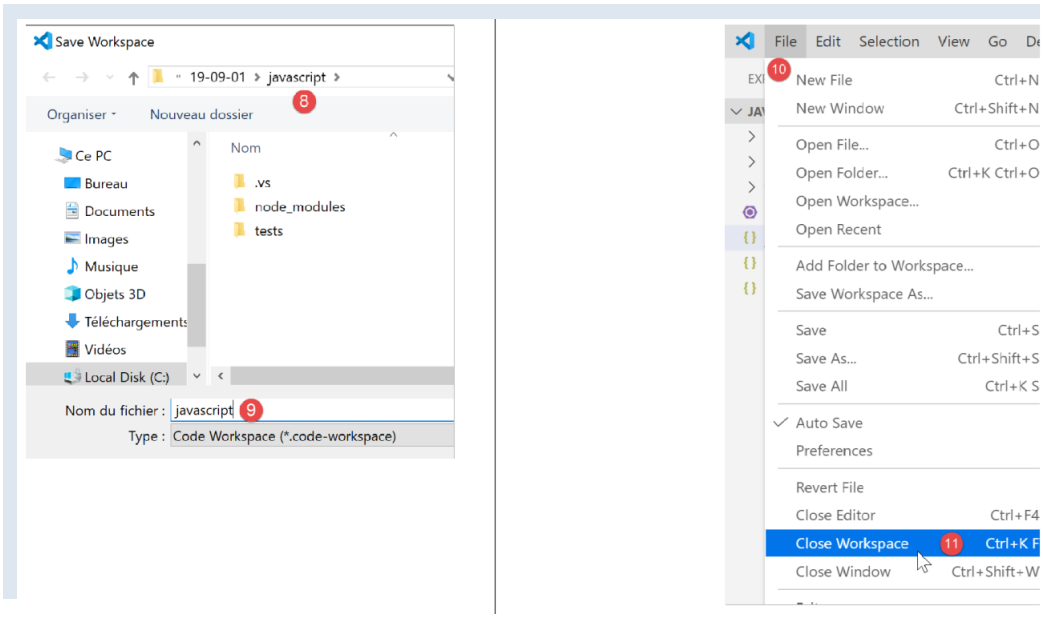
2.2.2.3 Quelques commandes [VSCode] utiles

- pour formater votre code, cliquez droit dessus **[1]** ;
- pour fermer les fenêtres ouvertes, cliquez droit sur leurs titres **[2-3]** ;

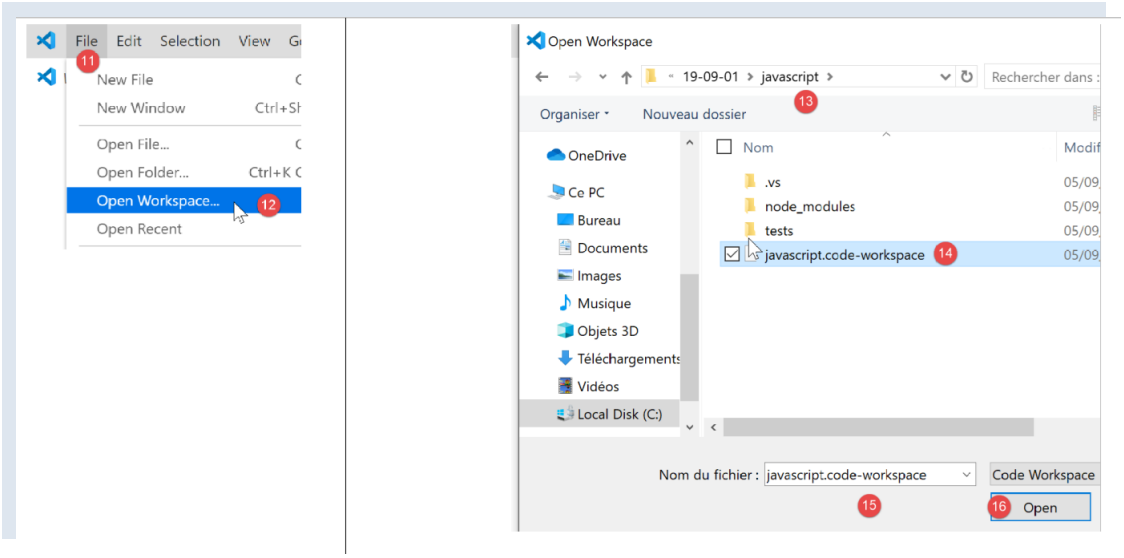


- pour afficher une fenêtre particulière **[4-5]** ;
- pour sauvegarder votre projet (Workspace) **[6-9]** ;
- pour sauvegarder un projet **[10-11]** ;

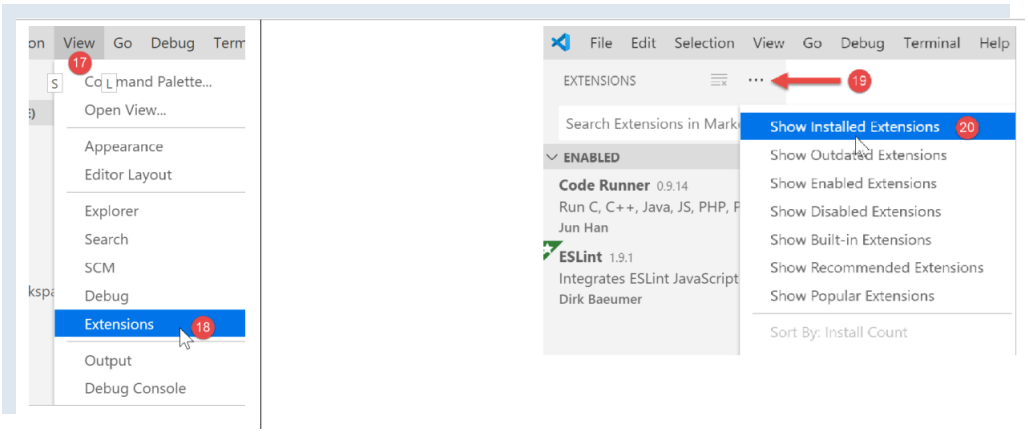




- pour ouvrir un projet **[11-16]** :



- voir les extensions installées **[19-20]** :

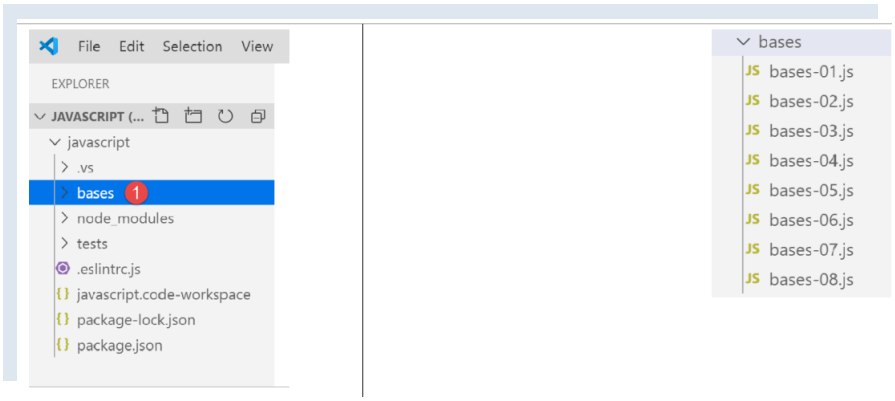


Nous avons désormais de bons outils pour développer en Javascript. Nous allons maintenant présenter ce langage à l'aide de courts extraits de code. Comme cette présentation se fait à la suite d'un cours PHP, nous ferons parfois des comparaisons entre ces deux langages pour signaler des différences entre eux.

2.3 Les bases de Javascript

Note : dans la suite, le terme **[Javascript]** désignera toujours la norme ECMAScript 6.

A l'intérieur du projet Javascript précédent, créez un dossier **[bases]**. Nous y mettrons les exemples de cette section :



2.3.1 script [bases-01]

Pour introduire les bases de PHP7, nous avions utilisé le code suivant(cf. paragraphe lien) :

```
1   <?php
2
3   // ceci est un commentaire
4   // variable utilisée sans avoir été déclarée
5   $nom = "dupont";
6   // un affichage écran
7   print "nom=$nom\n";
8   // un tableau avec des éléments de types différents
9   $tableau = array("un","deux",3,4);
10  // son nombre d'éléments
11  $n = count($tableau);
12  // une boucle
13  for ($i = 0; $i < $n; $i ++) {
14    print "tableau[$i]=$tableau[$i]\n";
15  }
16  // initialisation de 2 variables avec le contenu d'un tableau
17  list ($chaine1, $chaine2) = array("chaine1","chaine2");
18  // concaténation des 2 chaînes
19  $chaine3 = $chaine1 . $chaine2;
20  // affichage résultat
21  print "[$chaine1,$chaine2,$chaine3]\n";
22  // utilisation fonction
23  affiche($chaine1);
24  // le type d'une variable peut être connu
25  afficheType("n", $n);
26  afficheType("chaine1", $chaine1);
27  afficheType("tableau", $tableau);
28  // le type d'une variable peut changer en cours d'exécution
29  $n = "a changé";
30  afficheType("n", $n);
31  // une fonction peut rendre un résultat
32  $res1 = f1(4);
33  print "res1=$res1\n";
34  // une fonction peut rendre un tableau de valeurs
35  list ($res1, $res2, $res3) = f2();
36  print "(res1,res2,res3)=[$res1,$res2,$res3]\n";
37  // on aurait pu récupérer ces valeurs dans un tableau
38  $t = f2();
39  for ($i = 0; $i < count($t); $i ++) {
40    print "t[$i]=$t[$i]\n";
41  }
42  // des tests
43  for ($i = 0; $i < count($t); $i ++) {
44    // n'affiche que les chaînes
45    if (getType($t[$i]) === "string") {
```

```
46      print "t[$i]=$t[$i]\n";
47    }
48  }
49  // opérateurs de comparaison == et ===
50  if ("2" == 2) {
51    print "avec l'opérateur ==, la chaîne 2 est égale à l'entier 2\n";
52  } else {
53    print "avec l'opérateur ==, la chaîne 2 n'est pas égale à l'entier 2\n";
54  }
55  if ("2" === 2) {
56    print "avec l'opérateur ===, la chaîne 2 est égale à l'entier 2\n";
57  } else {
58    print "avec l'opérateur ===, la chaîne 2 n'est pas égale à l'entier 2\n";
59  }
60  // d'autres tests
61  for ($i = 0; $i < count($t); $i ++) {
62    // n'affiche que les entiers >10
63    if (getType($t[$i]) === "integer" and $t[$i] > 10) {
64      print "t[$i]=$t[$i]\n";
65    }
66  }
67  // une boucle while
68  // initialisation tableau (PHP 7)
69  $t = [8,5,0,- 2,3,4];
70  $i = 0;
71  $somme = 0;
72  while ($i < count($t) and $t[$i] > 0) {
73    print "t[$i]=$t[$i]\n";
74    $somme += $t[$i]; // $somme=$somme+$t[$i]
75    $i ++; // $i=$i+1
76  } // while
77  print "somme=$somme\n";
78
79  // notation d'une constante
80  const TAUX_TVA = 19.6;
81  print "taux de TVA=" . TAUX_TVA;
82
83  // arrêt du programme
84  exit();
85
86  // affiche
87  // ----------------------------------
88  function affiche($chaine) {
89    // affiche $chaine
90    print "chaine=$chaine\n";
91  }
92
93  // afficheType
94  // ----------------------------------
95  function afficheType($name, $variable) {
96    // affiche le type de $variable
97    print "type[variable $" . $name . "]=" . getType($variable) . "\n";
98  }
99
100 // ----------------------------------
101 function f1($param) {
102   // ajoute 10 à $param
103   return $param + 10;
104 }
105
106 // ----------------------------------
107 function f2() {
108   // rend 3 valeurs
109   return array("un",0,100);
110 }
```

Traduit en Javascript, cela donne le code suivant :

```
1.  'use strict';
2.  // ceci est un commentaire
3.  // constante
4.  const nom = "dupont";
5.  // un affichage écran
6.  console.log("nom : ", nom);
7.  // un tableau avec des éléments de types différents
8.  const tableau = ["un", "deux", 3, 4];
9.  // son nombre d'éléments
```

```
let n = tableau.length;
// une boucle
for (let i = 0; i < n; i++) {
  console.log("tableau[", i, "] = ", tableau[i]);
}
// initialisation de 2 variables avec le contenu d'un tableau
let [chaine1, chaine2] = ["chaine1", "chaine2"];
// concaténation des 2 chaînes
const chaine3 = chaine1 + chaine2;
// affichage résultat
console.log([chaine1, chaine2, chaine3]);
// utilisation fonction
affiche(chaine1);
// le type d'une variable peut être connu
afficheType("n", n);
afficheType("chaine1", chaine1);
afficheType("tableau", tableau);
// le type d'une variable peut changer en cours d'exécution
n = "a changé";
afficheType("n", n);
// une fonction peut rendre un résultat
let res1 = f1(4);
console.log("res1=", res1);
// une fonction peut rendre un tableau de valeurs
let res2, res3;
[res1, res2, res3] = f2();
console.log("(res1,res2,res3)=", [res1, res2, res3]);
// on aurait pu récupérer ces valeurs dans un tableau
let t = f2();
for (let i = 0; i < t.length; i++) {
  console.log("t[", i, "]=", t[i]);
}
// des tests
for (let i = 0; i < t.length; i++) {
  // n'affiche que les chaînes
  if (typeof (t[i]) === "string") {
    console.log("t[", i, "]=", t[i]);
  }
}
// opérateurs de comparaison == et ===
if ("2" == 2) {
  console.log("avec l'opérateur ==, la chaîne 2 est égale à l'entier 2");
} else {
  console.log("avec l'opérateur ==, la chaîne 2 n'est pas égale à l'entier 2");
}
if ("2" === 2) {
  console.log("avec l'opérateur ===, la chaîne 2 est égale à l'entier 2");
} else {
  console.log("avec l'opérateur ===, la chaîne 2 n'est pas égale à l'entier 2");
}
// d'autres tests
for (let i = 0; i < t.length; i++) {
  // n'affiche que les entiers >10
  if (typeof (t[i]) === "number" && Math.floor(t[i]) === t[i] && t[i] > 10) {
    console.log("t[", i, "]=", t[i]);
  }
}
// une boucle while
t = [8, 5, 0, -2, 3, 4];
let i = 0;
let somme = 0;
while (i < t.length && t[i] > 0) {
  console.log("t[", i, "]=", t[i]);
  somme += t[i];
  i++;
}
console.log("somme=", somme);

// arrêt du programme car il n'y a plus de code exécutable

//affiche
//---------------------------------
function affiche(chaine) {
  // affiche chaine
  console.log("chaine=", chaine);
}

```

```
87. //afficheType
88. //---------------------------------
89. function afficheType(name, variable) {
90.   // affiche le type de variable
91.   console.log("type[variable ", name, "]=", typeof (variable));
92. }
93.
94. //---------------------------------
95. function f1(param) {
96.   // ajoute 10 à param
97.   return param + 10;
98. }
99.
100.//---------------------------------
101.function f2() {
102.  // rend 3 valeurs
103.  return ["un", 0, 100];
104.}
```

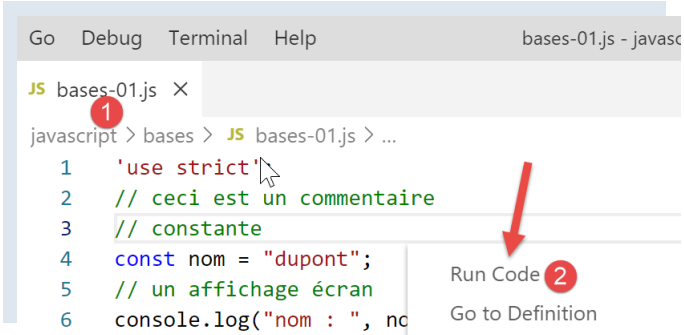
Commentons les différences entre les codes PHP et ECMAScript 6 avec la déclaration **[use strict]** (ligne 1) :

- la 1ère différence est qu'en ECMAScript **on déclare les variables** avec les mots clés suivants :
  - **[let]** pour déclarer une variable dont la valeur **peut changer** au cours de l'exécution du code ;
  - **[const]** pour déclarer une variable dont la valeur **ne va pas changer** (une constante donc) au cours de l'exécution du code ;
  - on peut également utiliser le mot clé **[var]** à la place de **[let]**. C'était le mot clé utilisé avec ECMAScript 5. Nous ne l'utiliserons pas dans ce cours ;
- ligne 6 : la méthode d'affichage **[console.log]** peut afficher toutes sortes de données : chaînes, nombres, booléens, tableaux, objets. La méthode PHP **[print]** ne sait pas afficher nativement des tableaux et objets. Dans l'expression **[console.log]**, **[console]** est un objet et **[log]** une méthode de cet objet ;
- ligne 8 : les tableaux Javascript sont des **objets référencés par un pointeur**. Lorsqu'on écrit :

```
const tableau = ["un", "deux", 3, 4];
```

  - la variable **[tableau]** est un **pointeur** sur le tableau littéral **["un", "deux", 3, 4]**. Modifier le contenu du tableau ne modifie pas son pointeur. Aussi un tableau sera-t-il le plus souvent déclaré avec le mot clé **[const]**. En PHP, un tableau n'est pas référencé par un pointeur. C'est une donnée littérale ;
- ligne 12 : la variable de boucle **[i]** est déclarée (let) dans la boucle. Le mot clé **[let]** respecte la portée de bloc (code entre accolades). Ainsi la variable **[i]** de la ligne 12 n'est-elle connue que dans la boucle ;
- ligne 18 : l'opérateur de concaténation de chaîne est l'opérateur + en Javascript, . en PHP. Une particularité de cet opérateur est qu'il a précédence sur l'opérateur + d'addition. Ainsi :
  - en PHP, '1' +2 donne le nombre 3 ;
  - en Javascript '1'+2 donne la chaîne '12' ;
- ligne 20 : **[console.log]** sait afficher des tableaux ;
- ligne 82 : en Javascript, il n'est pas possible d'indiquer le type des paramètres d'une fonction ;
- ligne 91 : l'opérateur **[typeof]** permet de connaître le type d'une donnée. Il y en a quatre : nombre, chaîne de caractères, booléen et objet. On notera qu'en Javascript on n'a pas de type **[integer]** ni de type **[tableau]**. Comme il a été dit, les tableaux sont manipulés via des pointeurs et tombe dans la catégorie des objets ;
- lignes 50-59 : comme en PHP, Javascript a deux opérateurs de comparaison, == et '===' avec la même signification qu'en PHP. ESLint signale le plus souvent l'opérateur == comme une erreur possible. On utilisera systématiquement l'opérateur '===' ;
- ligne 79 : on aurait pu mettre l'instruction **[return]** mais ESLint émet l'avertissement que **[return]** ne doit s'utiliser que dans une fonction ;

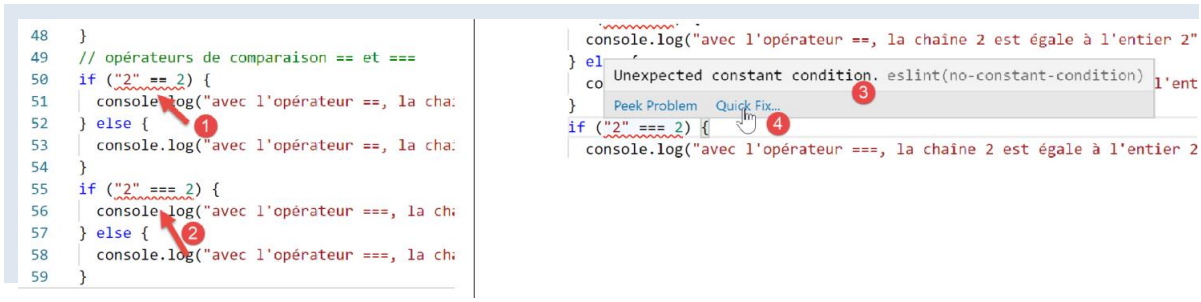
Exécutons ce code :



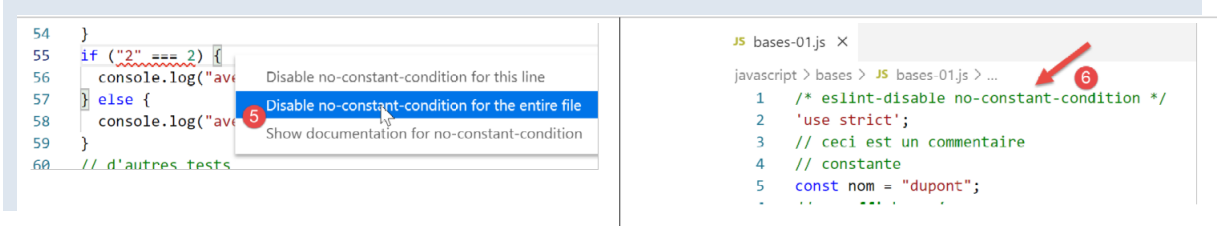
Les résultats de l'exécution :

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\bases\bases-01.js"
nom : dupont
tableau[ 0 ] = un
tableau[ 1 ] = deux
tableau[ 2 ] = 3
tableau[ 3 ] = 4
[ 'chaine1', 'chaine2', 'chaine1chaine2' ]
chaine= chaine1
type[variable n ]= number
type[variable chaine1 ]= string
type[variable tableau ]= object
type[variable n ]= string
res1= 14
(res1,res2,res3)= [ 'un', 0, 100 ]
t[ 0 ]= un
t[ 1 ]= 0
t[ 2 ]= 100
t[ 0 ]= un
avec l'opérateur ==, la chaîne 2 est égale à l'entier 2
avec l'opérateur ===, la chaîne 2 n'est pas égale à l'entier 2
t[ 2 ]= 100
t[ 0 ]= 8
t[ 1 ]= 5
somme= 13

[Done] exited with code=0 in 0.316 seconds
```

Dans le code écrit, ESLINT signale deux erreurs :



- en passant le curseur sur la ligne rouge de l'avertissement, on a le message d'erreur **[3]**. Ici ESLint ne comprend pas qu'on compare deux constantes. L'un des deux opérandes devrait être une variable ;
- en **[4]**, une option **[Quick Fix]** permet de lever l'avertissement si on décide de ne pas corriger l'erreur ;



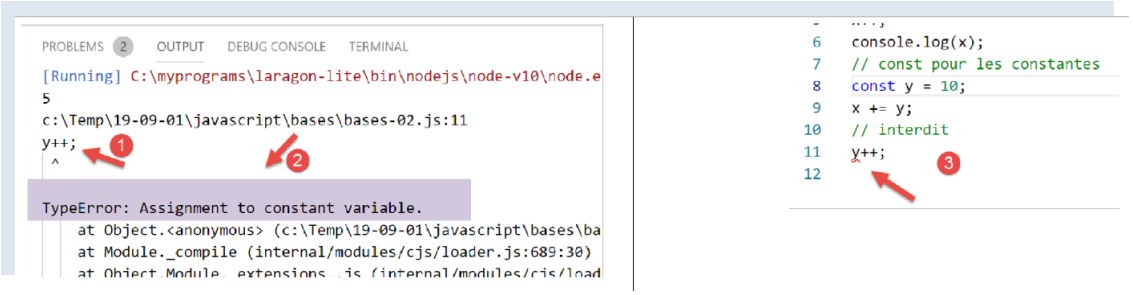
- en **[5]**, on a la possibilité de désactiver l'avertissement pour la ligne courante ou pour l'ensemble du fichier. C'est cette dernière option que nous choisissons ici . La ligne **[6]** est alors générée au début du fichier ;

2.3.2 script [bases-02]

Le script **[bases-02]** montre l'utilisation des mots clés **[let]** et **[const]** :

```
1.  'use strict';
2.  // pour initialiser une variable, on utilise let ou const
3.  // let pour les variables
4.  let x = 4;
5.  x++;
6.  console.log(x);
7.  // const pour les constantes
8.  const y = 10;
9.  x += y;
10. // interdit
11. y++;
```

- la ligne 11 provoque une erreur à l'exécution **[1-2]**. Elle est signalée par ESLint avant l'exécution **[3]** :



2.3.3 script [bases-03]

Le script **[bases-03]** examine la portée des variables en Javascript :

```
1.  'use strict';
2.  // portée des variables
3.  let count = 1;
4.  function doSomething() {
5.    // count est ici connu
6.    console.log("count=",count);
7.  }
8.  // appel
9.  doSomething();
```

- la variable **[count]** déclarée en-dehors de la fonction **[doSomething]** est pourtant connue dans cette fonction. C'est une différence fondamentale avec PHP ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\bases\bases-
    03.js"
2   count= 1
3
4   [Done] exited with code=0 in 0.3 seconds
```

2.3.4 script [bases-04]

Une variable locale cache une variable globale de même nom :

```
1.  'use strict';
2.  // portée des variables
3.  const count = 1;
4.  function doSomething() {
5.    // la variable locale cache la variable globale
6.    const count = 2;
7.    console.log("count inside function=",count);
```

```
8.  }
9.  // variable globale
10. console.log("count outside function=",count);
11. // variable locale
12. doSomething();
```

Exécution

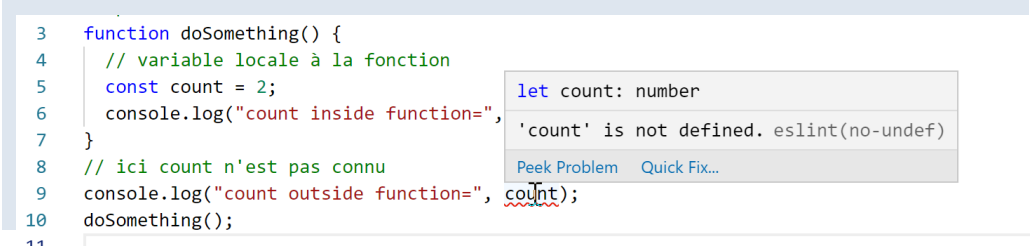
```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\bases\bases-
    04.js"
2   count outside function= 1
3   count inside function= 2
4
5   [Done] exited with code=0 in 0.246 seconds
```

2.3.5 script [bases-05]

Une variable définie dans une fonction n'est pas connue en-dehors de celle-ci :

```
1.  'use strict';
2.  // portée des variables
3.  function doSomething() {
4.    // variable locale à la fonction
5.    const count = 2;
6.    console.log("count inside function=", count);
7.  }
8.  // ici count n'est pas connu
9.  console.log("count outside function=", count);
10. doSomething();
```

ESLint déclare une erreur sur la ligne 9 :



2.3.6 script [bases-06]

Les mots clés **[let]** et **[const]** définissent des variables de portée **[bloc]** (code entre accolades) mais pas le mot clé **[var]** :

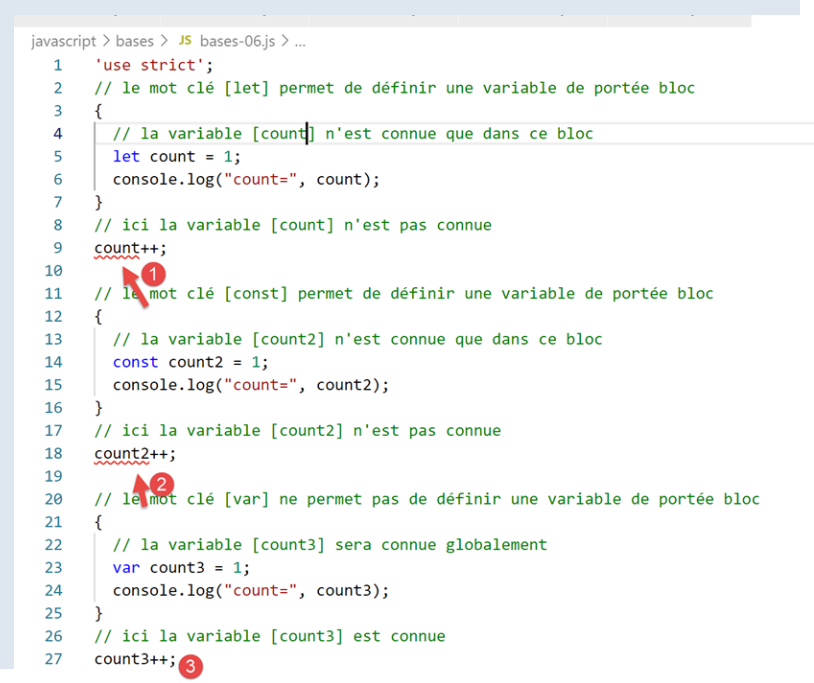
```
1.  'use strict';
2.  // le mot clé [let] permet de définir une variable de portée bloc
3.  {
4.    // la variable [count] n'est connue que dans ce bloc
5.    let count = 1;
6.    console.log("count=", count);
7.  }
8.  // ici la variable [count] n'est pas connue
9.  count++;
10.
11. // le mot clé [const] permet de définir une variable de portée bloc
12. {
13.   // la variable [count2] n'est connue que dans ce bloc
14.   const count2 = 1;
15.   console.log("count=", count2);
16. }
17. // ici la variable [count2] n'est pas connue
18. count2++;
19.
20. // le mot clé [var] ne permet pas de définir une variable de portée bloc
21. {
22.   // la variable [count3] sera connue globalement
23.   var count3 = 1;
24.   console.log("count=", count3);
25. }
26. // ici la variable [count3] est connue
```

```
27. count3++;
```

Commentaires

- ligne 5 : la variable **[count]** n'est connue que dans le bloc de code dans laquelle elle est déclarée (lignes 3-7) ;
- ligne 14 : la constante **[count2]** n'est connue que dans le bloc de code dans laquelle elle est déclarée (lignes 12-16) ;
- ligne 23 : la variable **[count3]** est connue en-dehors du bloc de code dans laquelle elle est déclarée (lignes 21-25) ;

ESLint déclare les erreurs suivantes :



Pour ces raisons de portée de bloc, nous n'utiliserons par la suite que les mots clés **[let]** et **[const]**.

2.3.7 script [bases-07]

Les types de données en Javascript :

```
1.  'use strict';
2.
3.  // type de données jS
4.  const var1 = 10;
5.  const var2 = "abc";
6.  const var3 = true;
7.  const var4 = [1, 2, 3];
8.  const var5 = {
9.    nom: 'axèle'
10. };
11. const var6 = function () {
12.   return +3;
13. }
14. // affichage des types
15. console.log("typeof(var1)=", typeof (var1));
16. console.log("typeof(var2)=", typeof (var2));
17. console.log("typeof(var3)=", typeof (var3));
18. console.log("typeof(var4)=", typeof (var4));
19. console.log("typeof(var5)=", typeof (var5));
20. console.log("typeof(var6)=", typeof (var6));
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\bases\bases-
    07.js"
2   typeof(var1)= number
3   typeof(var2)= string
```

```
4   typeof(var3)= boolean
5   typeof(var4)= object
6   typeof(var5)= object
7   typeof(var6)= function
8
9   [Done] exited with code=0 in 0.26 seconds
```

Commentaires

- ligne 7 (code) : un tableau est un objet. A ce titre **[var4]** est un pointeur vers le tableau, pas le tableau lui-même ;
- ligne 8 (code) : **[var5]** est un pointeur vers un objet littéral. On verra que les objets littéraux de Javascript ressemblent fort aux instances de classe de PHP. Eux-aussi sont référencés via des pointeurs ;
- ligne 11 (code) : une variable peut être de type **[fonction]** (ligne 7 des résultats) ;

2.3.8 script [bases-08]

Ce script montre les changements de type possibles en Javascript.

```
1.  'use strict';
2.
3.  // changements implicites de types
4.  // type -->bool
5.  console.log("---------------[Conversion implicite vers un booléen]-----------------------------");
6.  showBool("abcd");
7.  showBool("");
8.  showBool([1, 2, 3]);
9.  showBool([]);
10. showBool(null);
11. showBool(0.0);
12. showBool(0);
13. showBool(4.6);
14. showBool({});
15. showBool(undefined);
16.
17. function showBool(data) {
18.   // la conversion de data en booléen se fait automatiquement dans le test qui suit
19.   console.log("[data=", data, "], [type(data)]=", typeof (data), "[valeur booléenne(data)]=", data ? true :
     false);
20. }
21.
22. // changements implicites de type  vers un type numérique
23. console.log("---------------[Conversion implicite vers un nombre]-----------------------------");
24. showNumber("12");
25. showNumber("45.67");
26. showNumber("abcd");
27.
28. function showNumber(data) {
29.   // data + 1 ne marche pas car alors jS fait une concaténation de chaînes plutôt qu'une addition
30.   const nombre = data * 1;
31.   console.log("[data=", data, "], [type(data)]=", typeof (data), "[nombre]=", nombre, "[type(nombre)]=", ty
    peof (nombre));
32. }
33.
34. // changements explicites de types vers un booléen
35. console.log("---------------[Conversion explicite vers un booléen]-----------------------------");
36. showBool2("abcd");
37. showBool2("");
38. showBool2([1, 2, 3]);
39. showBool2([]);
40. showBool2(null);
41. showBool2(0.0);
42. showBool2(0);
43. showBool2(4.6);
44. showBool2({});
45. showBool2(undefined);
46.
47. function showBool2(data) {
48.   // la conversion de data en booléen se fait explicitement dans le test qui suit
49.   console.log("[", data, "], [type(data)]=", typeof (data), "[valeur booléenne(data)]=", Boolean(data));
50. }
51. // changements explicites de type vers Number
52. console.log("---------------[Conversion explicite vers un nombre]-----------------------------");
53. showNumber2("12.45");
54. showNumber2(67.8);
55. showNumber2(true);
```

```
56. showNumber2(null);
57.
58. function showNumber2(data) {
59.   const nombre = Number(data);
60.   console.log("[data=", data, "], [type(data)]=", typeof (data), "[nombre]=", nombre, "[type(nombre)]=", ty
    peof (nombre));
61. }
62.
63. // vers String
64. console.log("---------------[Conversion explicite vers un string]-----------------------------");
65. showString(5);
66. showString(6.7);
67. showString(false);
68. showString(null);
69.
70. function showString(data) {
71.   const chaîne = String(data);
72.   console.log("[data=", data, "], [type(data)]=", typeof (data), "[chaîne]=", chaîne, "[type(chaîne)]=", ty
    peof (chaîne));
73. }
74.
75. // qqs conversions implicites inattendues
76. console.log("---------------[Autres cas]-----------------------------");
77. const string1 = '1000.78';
78. // concaténation de chaînes par défaut
79. const data1 = string1 + 1.034;
80. console.log("data1=", data1, "type=", typeof (data1));
81. const data2 = 1.034 + string1;
82. console.log("data2=", data2, "type=", typeof (data2));
83. // conversion explicite vers nombre
84. const data3 = Number(string1) + 1.034;
85. console.log("data3=", data3, "type=", typeof (data3));
86. // true est converti en le nombre 1
87. const data4 = true * 1.18;
88. console.log("data4=", data4, "type=", typeof (data4));
89. // false est converti en le nombre 0
90. const data5 = false * 1.18;
91. console.log("data5=", data5, "type=", typeof (data5));
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Data\st-
    2019\dev\es6\javascript\bases\bases-08.js"
2   ---------------[Conversion implicite vers un booléen]-----------------------------
3   [data= abcd ], [type(data)]= string [valeur booléenne(data)]= true
4   [data= ], [type(data)]= string [valeur booléenne(data)]= false
5   [data= [ 1, 2, 3 ] ], [type(data)]= object [valeur booléenne(data)]= true
6   [data= [] ], [type(data)]= object [valeur booléenne(data)]= true
7   [data= null ], [type(data)]= object [valeur booléenne(data)]= false
8   [data= 0 ], [type(data)]= number [valeur booléenne(data)]= false
9   [data= 0 ], [type(data)]= number [valeur booléenne(data)]= false
10  [data= 4.6 ], [type(data)]= number [valeur booléenne(data)]= true
11  [data= {} ], [type(data)]= object [valeur booléenne(data)]= true
12  [data= undefined ], [type(data)]= undefined [valeur booléenne(data)]= false
13  ---------------[Conversion implicite vers un nombre]-----------------------------
14  [data= 12 ], [type(data)]= string [nombre]= 12 [type(nombre)]= number
15  [data= 45.67 ], [type(data)]= string [nombre]= 45.67 [type(nombre)]= number
16  [data= abcd ], [type(data)]= string [nombre]= NaN [type(nombre)]= number
17  ---------------[Conversion explicite vers un booléen]-----------------------------
18  [ abcd ], [type(data)]= string [valeur booléenne(data)]= true
19  [ ], [type(data)]= string [valeur booléenne(data)]= false
20  [ [ 1, 2, 3 ] ], [type(data)]= object [valeur booléenne(data)]= true
21  [ [] ], [type(data)]= object [valeur booléenne(data)]= true
22  [ null ], [type(data)]= object [valeur booléenne(data)]= false
23  [ 0 ], [type(data)]= number [valeur booléenne(data)]= false
24  [ 0 ], [type(data)]= number [valeur booléenne(data)]= false
25  [ 4.6 ], [type(data)]= number [valeur booléenne(data)]= true
26  [ {} ], [type(data)]= object [valeur booléenne(data)]= true
27  [ undefined ], [type(data)]= undefined [valeur booléenne(data)]= false
28  ---------------[Conversion explicite vers un nombre]-----------------------------
29  [data= 12.45 ], [type(data)]= string [nombre]= 12.45 [type(nombre)]= number
30  [data= 67.8 ], [type(data)]= number [nombre]= 67.8 [type(nombre)]= number
31  [data= true ], [type(data)]= boolean [nombre]= 1 [type(nombre)]= number
32  [data= null ], [type(data)]= object [nombre]= 0 [type(nombre)]= number
33  ---------------[Conversion explicite vers un string]-----------------------------
34  [data= 5 ], [type(data)]= number [chaîne]= 5 [type(chaîne)]= string
35  [data= 6.7 ], [type(data)]= number [chaîne]= 6.7 [type(chaîne)]= string
```

```
36  [data= false ], [type(data)]= boolean [chaîne]= false [type(chaîne)]= string
37  [data= null ], [type(data)]= object [chaîne]= null [type(chaîne)]= string
38  ---------------[Autres cas]-----------------------------
39  data1= 1000.781.034 type= string
40  data2= 1.0341000.78 type= string
41  data3= 1001.814 type= number
42  data4= 1.18 type= number
43  data5= 0 type= number
```

2.4 Les tableaux

2.4.1 script [tab-01]

Le script suivant illustre certaines caractéristiques des tableaux Javascript. Ceux-ci ressemblent aux tableaux PHP avec cependant une grande différence : ils sont manipulés via des pointeurs et sont considérés comme des objets.

```
1.  'use strict';
2.
3.  // un tableau est un objet manipulé via son adresse
4.  const tab1 = [1, 2, 3];
5.  // copie d'adresses
6.  const tab2 = tab1;
7.  // tab1 et tab2 pointent sur le même tableau
8.  console.log("tab1===tab2 :", tab1 === tab2);
9.  // on peut modifier le tableau en passant indifféremment par tab1 ou tab2
10. tab2[1] = 10;
11. console.log("tab1=", tab1);
12. console.log("tab2=", tab2);
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\tableaux\tab-01.js"
2.  tab1===tab2 : true
3.  tab1= [ 1, 10, 3 ]
4.  tab2= [ 1, 10, 3 ]
```

Commentaires

- la ligne 2 des résultats montre que **[tab1]** et **[tab2]** sont deux entités égales, en fait deux pointeurs égaux ;
- les lignes 4 et 5 des résultats montrent que ces deux pointeurs pointent sur le même tableau ;

2.4.2 script [tab-02]

Ce script montre que le tableau Javascript est différent des tableaux des langages compilés.

```
1.  'use strict';
2.
3.  // tableau
4.  const tab = [];
5.  console.log("tab=", tab, ", longueur=[", tab.length, "]");
6.  console.log("-------------------------------");
7.  // initialisation d'un élément
8.  tab[3] = 100;
9.  tab[1] = "huit";
10. // tableau
11. console.log("tab=", tab, ", longueur=[", tab.length, "]");
12. console.log("-------------------------------");
13. // toString
14. console.log("tab.toString=[", tab.toString(), "]");
15. console.log("-------------------------------");
16. // les clés du tableau sont ses indices
```

```
17. for (let key of tab.keys()) {
18.   console.log("clé=[", key, "], valeur=[", tab[key], "]");
19. }
20. console.log("------------------------------");
21. // les valeurs du tableau
22. for (let value of tab.values()) {
23.   console.log("valeur=[", value, "]");
24. }
```

- ligne 4 : un tableau vide ;
- ligne 8 : un tableau n'a pas de taille fixe. C'est simplement une suite d'éléments indexée par un n°. On peut initialiser l'élément n° 3 même si les éléments **[0, 1, 2]** ne sont pas encore définis ;
- lignes 16-24 : un tableau Javascript a un comportement analogue à celui du tableau PHP ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-
    01\javascript\tableaux\tab-02.js"
2   tab= [] , longueur=[ 0 ]
3   ------------------------------
4   tab= [ <1 empty item>, 'huit', <1 empty item>, 100 ] , longueur=[ 4 ]
5   ------------------------------
6   tab.toString=[ ,huit,,100 ]
7   ------------------------------
8   clé=[ 0 ], valeur=[ undefined ]
9   clé=[ 1 ], valeur=[ huit ]
10  clé=[ 2 ], valeur=[ undefined ]
11  clé=[ 3 ], valeur=[ 100 ]
12  ------------------------------
13  valeur=[ undefined ]
14  valeur=[ huit ]
15  valeur=[ undefined ]
16  valeur=[ 100 ]
```

2.4.3 script `[tab-03]`

Ce script montre différentes méthodes de l'objet **[tableau]**.

```
1.  'use strict';
2.  // un tableau peut contenir différents types de données
3.  const tab = [1, 2, "un", "deux", true, [10, 20], { prop1: 10, prop2: "abc" }];
4.  // console.log sait afficher le contenu d'un tableau
5.  show(1);
6.  console.log("tab=", tab);
7.  show(2);
8.  // parcours du tableau avec foreach
9.  tab.forEach(element => {
10.   console.log("élément=", element, typeof (element));
11. });
12. show("2b");
13. // une autre écriture pour faire la même chose
14. tab.forEach(function (element) {
15.   console.log("élément=", element, typeof (element));
16. });
17. show(3);
18. // parcours du tableau avec for
19. for (let i = 0; i < tab.length; i++) {
20.   console.log("i=", i, "tab[i]=", tab[i]);
21. }
22. show(4);
23. // modification tab[i]
24. tab[5] = [];
25. // affichage
26. console.log("tab=", tab);
27. show(5);
28. // on enlève le dernier élément du tableau
29. let element = tab.pop(tab);
30. console.log("élément=", element, "tab=", tab);
31. show(6);
32. // on ajoute un élément à la fin du tableau
33. tab.push('xyz');
34. console.log("tab=", tab);
35. show(7);
36. // on ajoute un élément au début du tableau
37. tab.unshift(1000);
```

```
38. console.log("tab=", tab);
39. show(8);
40. // on enlève le 1er élément
41. element = tab.shift();
42. console.log("élément=", element, "tab=", tab);
43. show(9);
44. // on enlève l'élément n° 2 du tableau
45. element = tab.splice(2, 1);
46. console.log("élément=", element, "tab=", tab);
47. show(10);
48. // on enlève du tableau deux éléments à partir de l'élément n° 1
49. element = tab.splice(1, 2);
50. console.log("élément=", element, "tab=", tab);
51.
52. // fonction
53. function show(param) {
54.   console.log("[", param, ":::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: ]");
55. }
```

Commentaires

- la différence entre tableaux PHP et tableaux Javascript est illustrée par les lignes 3 et 24 :
  - la ligne 3 déclare la variable **[tab]** comme une constante ;
  - la ligne 24 modifie l'élément tab**[5]** ;
    - Ligne 3, c'est le pointeur qui pointe sur le tableau qui est déclaré constant, ce n'est pas le tableau lui-même. Celui-ci peut être modifié.
- lignes 14-16 : le tableau **[tab]** est parcouru à l'aide d'une méthode **[forEach]** du tableau **[tab]**. Cette méthode reçoit en paramètre la définition d'une fonction, qu'on pourrait appeler une fonction littérale. Cette fonction paramètre reçoit un paramètre : l'élément courant du tableau **[tab]**. La fonction est appelée pour chaque élément du tableau. Ce type d'écriture est courant en Javascript ;
- lignes 9-10 : on utilise une autre syntaxe pour définir la fonction paramètre. Au lieu d'écrire :
  - **function(p1, p2, …, pn){….}**
    - on écrit :
  - **(p1,p2, ..,pn)=>{….}**. On appelle cela, la notation « flèche » ou « arrow » ;
- le reste du code est expliqué par les commentaires ;

De ce script, on notera que :

- un tableau est un objet référencé par un pointeur ;
- que cet objet a des méthodes **[forEach, pop, push, shift, unshift]** ;

2.4.4 script [tab-04]

Ce script présente d'autres méthodes des objets tableau.

```
1.  'use strict';
2.
3.  // méthode de manipulation de tableaux
4.
5.  // un tableau
6.  const tab = [];
7.  for (let i = 0; i < 10; i++) {
8.    tab[i] = i * 10;
9.  }
10. // affichage
11. console.log("tab=", tab);
12. // map
13. const tab2 = tab.map(element => {
14.   return { prop1: element, prop2: element * element }
15. });
16. // affichage
17. console.log("tab=", tab);
18. console.log("tab2=", tab2);
19. // reduce sans valeur initiale
20. const somme = tab.reduce((accumulator, currentValue) => accumulator + currentValue);
21. console.log("somme tab=", somme);
22. // reduce avec valeur initiale 10
23. const somme2 = tab.reduce((accumulator, currentValue) => accumulator + currentValue, 10);
24. console.log("somme2 tab=", somme2);
25. // filter
26. const tab4 = tab.filter((element) => {
27.   if (element > 50) {
```

```
28.     return element;
29.   }
30. });
31. console.log("tab4=", tab4);
32. // find
33. const element1 = tab.find((element) => (element > 20));
34. console.log("élément1=", element1);
35. // findIndex
36. const index1 = tab.findIndex((element) => (element === 20));
37. console.log("index1 20=", index1);
38. // indexOf
39. const index2 = tab.indexOf(30);
40. console.log("index2 30=", index2);
41. const index3 = tab.indexOf(31);
42. console.log("index3 31=", index3);
43. // lastIndexOf
44. const index4 = [4, 5, 4, 2].lastIndexOf(4);
45. console.log("index4 4=", index4);
46. // sort
47. const tab5 = [4, 5, 4, 2].sort();
48. console.log("tab5=", tab5);
49. // sort inverse
50. const tab6 = [4, 5, 4, 2].sort((e1, e2) => {
51.   if (e1 > e2) {
52.     return -1;
53.   }
54.   else if (e1 === e2) {
55.     return 0;
56.   } else {
57.     return +1;
58.   }
59. });
60. console.log("tab6=", tab6);
```

Commentaires

- lignes 13-15 : la méthode **[map]** admet une fonction de transformation comme paramètre. Celle-ci est appelée de façon répétée pour chaque élément du tableau. Elle est chargée de transformer celui-ci en autre chose, ici un objet avec les propriétés **[prop1, prop2]**. La méthode **[map]** rend un nouveau tableau. L'ancien n'est pas modifié :

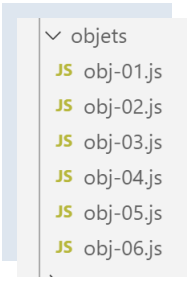
```
tab= [ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
tab2= [ { prop1: 0, prop2: 0 },
{ prop1: 10, prop2: 100 },
{ prop1: 20, prop2: 400 },
{ prop1: 30, prop2: 900 },
{ prop1: 40, prop2: 1600 },
{ prop1: 50, prop2: 2500 },
{ prop1: 60, prop2: 3600 },
{ prop1: 70, prop2: 4900 },
{ prop1: 80, prop2: 6400 },
{ prop1: 90, prop2: 8100 } ]
```

- ligne 20 : la méthode **[reduce]** admet pour paramètre une fonction à deux paramètres appelée de façon répétée pour chaque élément du tableau. Cette fonction admet deux paramètres :
  - **[currentValue]** est l'élément courant du tableau ;
  - **[accumulator]** est le dernier résultat obtenu par la fonction. Si aucune valeur initiale n'est prévue pour cet accumulateur, alors celle-ci sera 0 ;
  - la 1ère fois que la fonction d'accumulation est appelée, elle rend **[0+tab[0]]**. C'est cette valeur qui est affectée à l'accumulateur ;
  - la seconde fois, elle rend accumulateur+tab**[1]**, donc tab**[0]**+tab**[1]** ;
  - la troisième fois, elle rend accumulateur+tab**[2]**, donc tab**[0]**+tab**[1]**+tab**[2]** ;
  - etc. Au final, l'accumulateur représentera la somme de tous les éléments du tableau **[tab]** ;
- ligne 26 : la méthode **[filter]** a pour paramètre une fonction de filtrage. Celle-ci est appelée de façon répétée pour chaque élément du tableau et reçoit celui-ci en paramètre. Elle doit rendre :
  - **[true]** si l'élément doit être gardé ;
  - **[false]** sinon ;
- ligne 33 : la méthode **[find]** a pour paramètre une fonction de recherche. Celle-ci est appelée de façon répétée pour chaque élément du tableau et reçoit celui-ci en paramètre. Elle doit rendre **[true]** si l'élément reçu satisfait le critère de recherche. Celle-ci s'arrête alors. La méthode **[find]** rend donc 0 ou 1 élément ;
- ligne 36 : la méthode **[findIndex]** fonctionne comme la méthode **[find]** mais au lieu de rendre l'élément trouvé, elle rend son index dans le tableau ;

- lignes 39, 41, la méthode **[indexOf(valeur)]** recherche **[valeur]** dans le tableau et rend son index, ou -1 s'il n'est pas trouvé ;
- ligne 44 : la méthode **[lastIndexOf(valeur)]** fonctionne comme la méthode **[indexOf(valeur)]** mais commence sa recherche par la fin du tableau ;
- ligne 47 : la méthode **[sort]** sans paramètres rend un tableau trié dans l'ordre naturel (nombres, chaînes) ;
- ligne 50 : lorsque l'ordre naturel ne convient pas, il faut passer à la méthode **[sort]** une fonction à deux paramètres (e1,e2) qui rend :
  - +1 si e1 doit être classé après e2 ;
  - -1 si e1 doit être classé avant e2 ;
  - 0 si les deux éléments doivent avoir le même classement ;
    - La fonction passée en paramètre à la méthode **[sort]** est appelée de façon répétée par celle-ci pour comparer deux éléments du tableau ;

2.5 Les objets littéraux

Nous appelons ici 'objets littéraux' des objets définis littéralement dans le code. Javascript a la notion de classe, et d'objet instance de classe. Ce n'est donc pas ce type d'objet dont nous parlons maintenant.



2.5.1 script [obj-01]

Nous présentons ici les premières propriétés des objets littéraux. La principale est que l'objet est manipulé via un pointeur.

```
1.  'use strict';
2.  // un objet vide
3.  const obj1 = {};
4.  // on peut créer dynamiquement les propriétés de l'objet
5.  obj1.prop1 = "abcd";
6.  console.log('obj1=', obj1);
7.  // autre propriété
8.  obj1.prop2 = [1, 2, 3];
9.  console.log("obj1=", obj1);
10. // autre propriété avec une notation différente
11. obj1['prop3'] = true;
12. console.log("obj1=", obj1);
13. // obj1 est une référence sur l'objet (pointeur), pas l'objet lui-même
14. const obj2 = obj1;
15. // obj2 et obj1 pointent sur le même objet
16. obj2.prop1 = "xyzt";
17. console.log("obj1=", obj1);
18. console.log("obj2=", obj2);
19. // les propriété peuvent être des variables
20. const var1 = 'prop1';
21. console.log('prop1=', obj1[var1]);
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\objets\obj-
    01.js"
2   obj1=[object Object]
3   obj1= { prop1: 'abcd', prop2: [ 1, 2, 3 ] }
4   obj1= { prop1: 'abcd', prop2: [ 1, 2, 3 ], prop3: true }
5   obj1= { prop1: 'xyzt', prop2: [ 1, 2, 3 ], prop3: true }
6   obj2= { prop1: 'xyzt', prop2: [ 1, 2, 3 ], prop3: true }
7   prop1= xyzt
```

Commentaires

- ligne 3 du code : un objet est manipulé via un pointeur. Donc **[obj1]** est un pointeur. Modifier l'objet pointé ne modifie pas le pointeur **[obj1]**. C'est pourquoi, comme pour les tableaux, une référence d'objet est déclarée avec le mot clé **[const]** ;
- ligne 6 du code : comme pour les tableaux, **[console.log]** sait afficher des objets ;

- ligne 11 du code : obj1.prop3 peut être réécrit obj1.**['prop3']**. Cette dernière notation est utile lorsque 'prop3' est en fait une variable (lignes 20-21) ;
- lignes 13-18 du code : montrent que l'instruction **[obj2=obj1]** est une copie de référence d'objet et non de l'objet lui-même ;

2.5.2 script [obj-02]

Ce script montre que les propriétés d'un objet peuvent avoir pour valeur un objet. On a alors des objets multi-niveaux. On introduit également l'objet global **[JSON]** qui permet de faire des conversions objet ↔ chaîne de caractères.

```
1.  'use strict';
2.  // un objet à plusieurs niveaux
3.  const personne = {
4.    prénom: "martin",
5.    âge: 12,
6.    père: {
7.      prénom: "paul",
8.      âge: 45
9.    },
10.   mère: {
11.     prénom: "micheline",
12.     âge: 42
13.   }
14. }
15. // accès aux propriétés
16. console.log("prénom personne=", personne.prénom);
17. console.log("prénom mère=", personne.mère.prénom);
18. personne.mère.âge = 40;
19. console.log("âge mère=", personne.mère.âge);
20. // console.log sait afficher des objets
21. console.log("personne=", personne);
22. console.log("mère=", personne.mère);
23. // on peut aussi afficher la chaîne jSON de l'objet
24. let json = JSON.stringify(personne);
25. console.log("jSON=", json);
26. // on peut relire le jSON
27. let personne2 = JSON.parse(json);
28. console.log("père=", personne2.père);
```

Commentaires

- ligne 24 : transformation d'un objet Javascript en chaîne jSON ;
- ligne 27 : transformation d'une chaîne jSON en objet Javascript ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\objets\obj-
    02.js"
2   prénom personne= martin
3   prénom mère= micheline
4   âge mère= 40
5   personne= { 'prénom': 'martin',
6   'âge': 12,
7   'père': { 'prénom': 'paul', 'âge': 45 },
8   'mère': { 'prénom': 'micheline', 'âge': 40 } }
9   mère= { 'prénom': 'micheline', 'âge': 40 }
10  jSON= {"prénom":"martin","âge":12,"père":{"prénom":"paul","âge":45},"mère":{"prénom":"micheline","âge":40}}
11  père= { 'prénom': 'paul', 'âge': 45 }
```

Commentaires

- ligne 10 : dans une chaîne jSON, les propriétés sont obligatoirement entourées de guillemets ainsi que les valeurs de type chaîne de caractères ;

2.5.3 script [obj-03]

Ce script introduit la notion de getter / setter d'une propriété d'un objet :

```
1.  'use strict';
2.  // getters et setters d'un objet
3.  const personne = {
4.    // getter
5.    get nom() {
```

```
6.      console.log("getter nom");
7.      return this._nom;
8.    },
9.    // setter
10.   set nom(unNom) {
11.     console.log("setter nom");
12.     this._nom = unNom;
13.   }
14. };
15. // setter
16. personne.nom = "Hercule";
17. // getter
18. console.log(personne.nom);
19. // l'objet lui-même
20. console.log("personne=", personne);
21. // ça n'empêche pas d'accéder à la propriété [_nom] directement
22. personne._nom = "xyz";
23. console.log("personne=", personne);
```

Commentaires

- lignes 5-7 : définition d'un **[getter]**, une fonction qui rend généralement la valeur d'une propriété de l'objet mais qui en fait peut rendre n'importe quoi. Le mot clé **[function]** est remplacé par le mot clé **[get]** ;
- ligne 7 : le getter rend la propriété **[_nom]**. On voit que celle-ci n'a pas besoin d'être déclarée ;
- lignes 10-13 : définition d'un **[setter]**, une fonction qui affecte généralement la valeur reçue à une propriété de l'objet mais qui en fait peut faire n'importe quoi. Le mot clé **[function]** est remplacé par le mot clé **[set]**. Le **[setter]** peut être utilisé pour vérifier la validité de la valeur passée en paramètre au **[setter]** ;
- ligne 16 : la fonction **[set nom]** va être appelée implicitement ;
- ligne 18 : la fonction **[get nom]** va être appelée implicitement ;
- ligne 22 : montre que l'utilisation des getter / setter dépend de la bonne volonté du développeur. Si celui-ci connaît le nom de la propriété gérée par ceux-ci, il peut y accéder directement ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\objets\obj-
    03.js"
2   setter nom
3   getter nom
4   Hercule
5   personne= { nom: [Getter/Setter], _nom: 'Hercule' }
6   personne= { nom: [Getter/Setter], _nom: 'xyz' }
```

On notera lignes 5-6, que **[console.log]** affiche également les propriétés qui sont des fonctions.

2.5.4 script [obj-04]

Ce script montre trois façons d'écrire les noms des propriétés d'un objet et deux façons d'y accéder.

```
1.  'use strict';
2.  // les noms des propriétés d'un objet  peuvent être littéraux [nom], être entourés d'apostrophes ['nom']
3.  // ou de guillements ["nom"]
4.
5.  // littéraux
6.  const obj1 = {
7.    nom: "martin",
8.    prénom: "jean"
9.  };
10. console.log("prénom=", obj1.prénom);
11.
12. // entourés d'apostrophes
13. const obj2 = {
14.   'nom': "martin",
15.   'prénom': "jean"
16. };
17. console.log("nom=", obj2.nom);
18.
19. // entourés de guillemets
20. const obj3 = {
21.   "nom": "martin",
22.   "prénom": "jean"
23. };
24.
25. // deux syntaxes possibles pour accéder à la propriété [nom]
```

```
26. console.log("nom=", obj3.nom);
27. console.log("nom=", obj3['nom']);
28.
29. // notation raccourcie équivalente à {obj1:obj1, obj2:obj2}
30. const obj4 = {
31.   obj1, obj2
32. }
33.
34. console.log("obj4=", obj4)
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\objets\obj-
    04.js"
2   prénom= jean
3   nom= martin
4   nom= martin
5   nom= martin
```

2.5.5 script `[obj-05]`

Le script montre que les propriétés d'un objet littéral peuvent être des fonctions. On est alors très proche de l'objet instance de classe, où on a des propriétés et des méthodes.

```
1.  'use strict';
2.
3.  // un objet peut avoir des propriétés de type [function]
4.  const personne = {
5.    // propriétés
6.    prénom: "martin",
7.    âge: 12,
8.    père: {
9.      prénom: "paul",
10.     âge: 45
11.   },
12.   mère: {
13.     prénom: "micheline",
14.     âge: 42
15.   },
16.   // méthode
17.   toString: function () {
18.     return JSON.stringify(this);
19.   }
20. }
21.
22. // usage
23. console.log("personne=", personne);
24. console.log("personne.toString=", personne.toString());
```

- lignes 17-19 : une méthode interne à l'objet. Dans celle-ci, on accède aux propriétés de l'objet via le mot clé **`[this]`** (ligne 18). **`[this]`** désigne l'objet lui-même, **`[this.prénom]`**, la propriété **`[prénom]`** de celui-ci ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\objets\obj-
    05.js"
2   personne= { 'prénom': 'martin',
3   'âge': 12,
4   'père': { 'prénom': 'paul', 'âge': 45 },
5   'mère': { 'prénom': 'micheline', 'âge': 42 },
6   toString: [Function: toString] }
7   personne.toString=
    {"prénom":"martin","âge":12,"père":{"prénom":"paul","âge":45},"mère":{"prénom":"micheline","âge":42}}
```

2.5.6 script `[obj-06]`

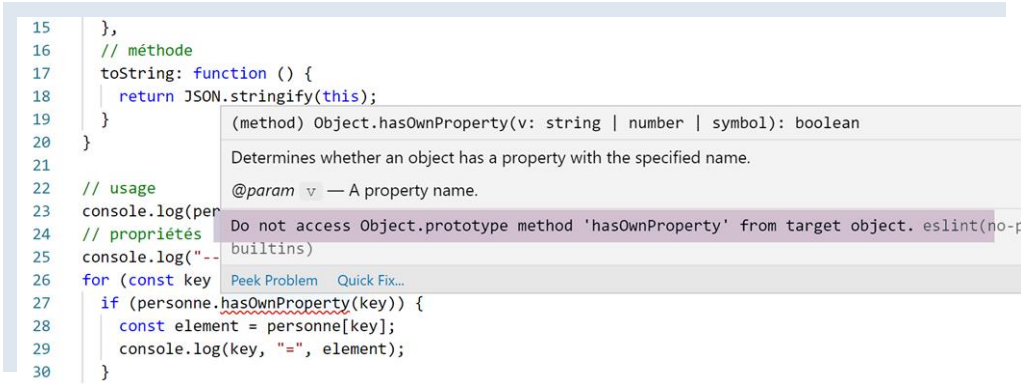
Ce script montre comment avoir accès aux propriétés d'un objet lorsqu'on ne connaît pas a priori le nom de celles-ci.

```
1.  'use strict';
2.
3.  // un objet peut avoir des propriétés de type [function]
4.  let personne = {
5.    // propriétés
6.    prénom: "martin",
```

```
7.    âge: 12,
8.    père: {
9.      prénom: "paul",
10.     âge: 45
11.   },
12.   mère: {
13.     prénom: "micheline",
14.     âge: 42
15.   },
16.   // méthode
17.   toString: function () {
18.     return JSON.stringify(this);
19.   }
20. }
21.
22. // usage
23. console.log(personne);
24. // propriétés
25. console.log("-----------------------");
26. for (const key in personne) {
27.   // eslint-disable-next-line no-prototype-builtins
28.   if (personne.hasOwnProperty(key)) {
29.     const element = personne[key];
30.     console.log(key, "=", element);
31.   }
32. }
33. // pour échapper à l'avertissement eslint (1)
34. console.log("-----------------------");
35. for (const key in personne) {
36.   if (Object.prototype.hasOwnProperty.call(personne, key)) {
37.     const element = personne[key];
38.     console.log(key, "=", element);
39.   }
40. }
41. // pour échapper à l'avertissement eslint (2)
42. console.log("-----------------------");
43. for (const key in personne) {
44.   // eslint-disable-next-line no-prototype-builtins
45.   if (personne.hasOwnProperty(key)) {
46.     const element = personne[key];
47.     console.log(key, "=", element);
48.   }
49. }
```

Commentaires

- lignes 26-32 : le code qui permet d'avoir la liste des propriétés, sans les méthodes, d'un objet. Ce code fait l'objet d'un avertissement d'ESLint :



- lignes 32-39 : le code qui permet d'échapper à l'avertissement d'ESLint. On passe par le prototype de la classe **`[Object]`** ;
- lignes 41-47 : ou bien on se contente de désactiver l'avertissement (ligne 43) ;

2.5.7 script `[obj-07]`

Le script **`[obj-07]`** montre la possibilité de déstructurer un objet :

```
1.
2.  'use strict';
3.  // déstructuration
4.
5.  // littéraux
6.  const obj1 = {
7.    nom: "martin",
8.    prénom: "jean"
9.  };
10.
11. // déstructuration obj1 dans variables [n,p]
12. const { nom: n, prénom: p } = obj1;
13. console.log("n=", n, "p=", p);
14.
15. // déstructuration obj1 dans variables [n2,p2]
16. function f({ nom: n2, prénom: p2 }) {
17.   console.log("f-n2=", n2, "f-p2=", p2);
18. }
19. f(obj1);
20.
21. // déstructuration obj1 dans variables [nom,prénom]
22. function g({ nom: nom, prénom: prénom }) {
23.   console.log("g-nom=", nom, "g-prénom=", prénom);
24. }
25. g(obj1);
26.
27. // déstructuration obj1 dans variables [nom,prénom]
28. // avec notation raccoucie équivalente à h({nom:nom,prénom:prénom})
29. function h({ nom, prénom }) {
30.   console.log("h-nom=", nom, "h-prénom=", prénom);
31. }
32. h(obj1);
```

Commentaires

- ligne 12 : ce sont les accolades {} qui permettent la déstructuration. La syntaxe

  ```
  const { nom: n, prénom: p } = obj1
  ```

  crée deux variables **[n]** et **[p]** et est équivalente à :

  ```
  const n = obj1.nom
  const p = obj1.prénom
  ```

  La déclaration pourrait se lire de la façon suivante :

  ```
  const { nom => n, prénom => p } = obj1
  ```

  pour rappeler que les valeurs des attributs **[nom, prénom]** vont dans les variables **[n, p]** ;

- l'opération de déstructuration se répète aux lignes 16, 22 et 29. A chaque fois, c'est la présence des accolades {} qui indique qu'il va y avoir déstructuration d'un objet dans des variables ;
- la ligne 29 peut être déconcertante. C'est un raccourci pour la notation :

  ```
  function h({ nom : nom, prénom : prénom })
  ```

Les résultats de l'exécution sont les suivants :

```
1   n= martin p= jean
2   f-n2= martin f-p2= jean
3   g-nom= martin g-prénom= jean
4   h-nom= martin h-prénom= jean
```

2.5.8 script [obj-08]

Le script **[obj-08]** montre comment obtenir une copie d'un objet :

```
1.  'use strict'
2.
3.  // clônage d'objets
4.  const obj1 = {
5.    nom: "martin",
```

```
6.    prénom: "jean"
7.  };
8.
9.  // clône de obj1 avec l'opérateur de spread
10. const obj2 = { ...obj1 }
11.
12. // vérifications
13. // obj2 pointe sur une copie de obj1
14. console.log("obj2===obj1 :", obj1 === obj2)
15. console.log("obj2=", obj2)
```

- ligne 10 : l'opération de copie de l'objet **[obj1]**. L'opérateur ... est appelé opérateur de spread ;

Les résultats de l'exécution sont les suivants :

```
1   obj2===obj1 : false
2   obj2= { nom: 'martin', 'prénom': 'jean' }
```

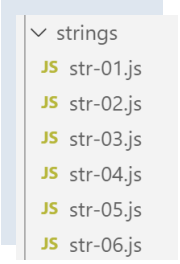
- ligne 1 : montre que les références **[obj1]** et **[obj2]** ne pointent pas sur le même objet ;
- ligne 2 : montre que l'objet pointé par **[obj2]** est une copie de l'objet pointé par **[obj1]** ;

2.5.9 Conclusion

Les scripts de cette section ont montré que l'objet littéral de Javascript est proche de l'objet instance de classe des langages à objets. On peut y définir propriétés, méthodes et getters / setters. C'est un objet dynamique dont on peut définir les propriétés à l'exécution. Il se comporte alors comme un dictionnaire dont les éléments peuvent être de tout type et notamment de type **[fonction]**.

2.6 Les chaînes de caractères

Les chaînes de caractères de Javascript sont très semblables à celles de PHP.



2.6.1 script [str-01]

La première chose à comprendre est qu'une fois une chaîne créée, elle n'est plus modifiable. On dispose de nombreuses méthodes pour produire une nouvelle chaîne à partir de la chaîne initiale mais celle-ci reste toujours inchangée. Par ailleurs, une chaîne de caractères peut être de deux types :

- **[string]** lorsqu'elle est initialisée avec une chaîne littérale ;
- **[object]** lorsqu'elle est créée comme instance de la classe **[String]** ;

```
1.  'use strict';
2.
3.  // les chaînes de caractères sont en lecture seule (on ne peut pas les modifier)
4.
5.  // une chaîne
6.  const chaîne1 = "abcd ";
7.  // type
8.  console.log("typeof(chaîne1)=", typeof (chaîne1));
9.  // caractère n° 2
10. console.log("chaîne1[2]=", chaîne1[2]);
11. // provoque une erreur
12. chaîne1[2] = "0";
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Temp\19-09-
    01\javascript\strings\str-01.js"
2   typeof(chaîne1)= string
```

```
3   chaîne1[2]= c
4   c:\Temp\19-09-01\javascript\strings\str-01.js:1
5   TypeError: Cannot assign to read only property '2' of string 'abcd '
6   at Object.<anonymous> (c:\Temp\19-09-01\javascript\strings\str-01.js:12:12)
7   at Generator.next (<anonymous>)
```

2.6.2 script [str-02]

Ce script montre qu'on peut construire une chaîne de caractères de deux façons.

```
1.  'use strict';
2.
3.  // les chaînes de caractères peuvent être de deux types
4.
5.  // une chaîne littérale
6.  const chaîne1 = "abcd ";
7.  // type
8.  console.log("typeof(chaîne1)=", typeof (chaîne1));
9.  // instance de String
10. const chaîne2 = new String("xyzt");
11. // type
12. console.log("typeof(chaîne2)=", typeof (chaîne2));
13. // autre écriture (sans new)
14. const chaîne3 = String("12 34");
15. // type
16. console.log("typeof(chaîne3)=", typeof (chaîne3));
17. // le type [string] et le type [object] offrent les mêmes méthodes, celles de la classe String
18. console.log("chaîne1.length=", chaîne1.length);
19. console.log("chaîne2.length=", chaîne2.length);
```

Commentaires

- ligne 6 : la méthode usuelle de définition d'une chaîne. **[chaîne1]** sera de type **[string]** ;
- ligne 10 : on peut construire une chaîne à l'aide du constructeur de la classe **[String]**. **[chaîne2]** sera de type **[object]** ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\strings\str-
    02.js"
2   typeof(chaîne1)= string
3   typeof(chaîne2)= object
4   typeof(chaîne3)= string
5   chaîne1.length= 5
6   chaîne2.length= 4
```

Le type **[string]** bénéficie des méthodes de la classe **[String]**.

2.6.3 script [str-03]

Ce script montre une chaîne particulière avec interpolation de variables.

```
1   'use strict';
2
3   // chaine
4   const chaîne = "Introduction à Javascript par l'exemple";
5   // chaîne avec interpolation de variables
6   const str = `[${chaîne}].substr(3, 2)=` + chaîne.substr(3, 2)
7   console.log(str);
```

Commentaires

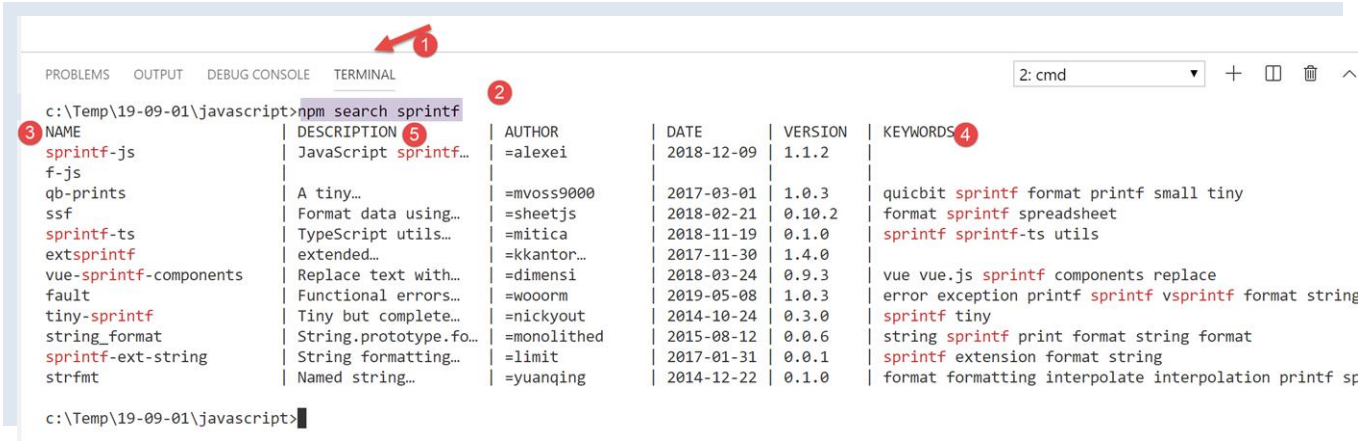
- ligne 6 : il est possible d'avoir des chaînes de caractères contenant des expression ${variable} qui sont remplacées par la valeur de la variable. On est là dans la même logique que les variables $ dans les chaînes de caractères PHP. On notera la notation d'une telle chaîne : elle est entourée de « backstick » ou apostrophe inverse (AltGr-7 sur un clavier français) ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Temp\19-09-
    01\javascript\strings\tempCodeRunnerFile.js"
2   [Introduction à Javascript par l'exemple].substr(3, 2)=ro
```

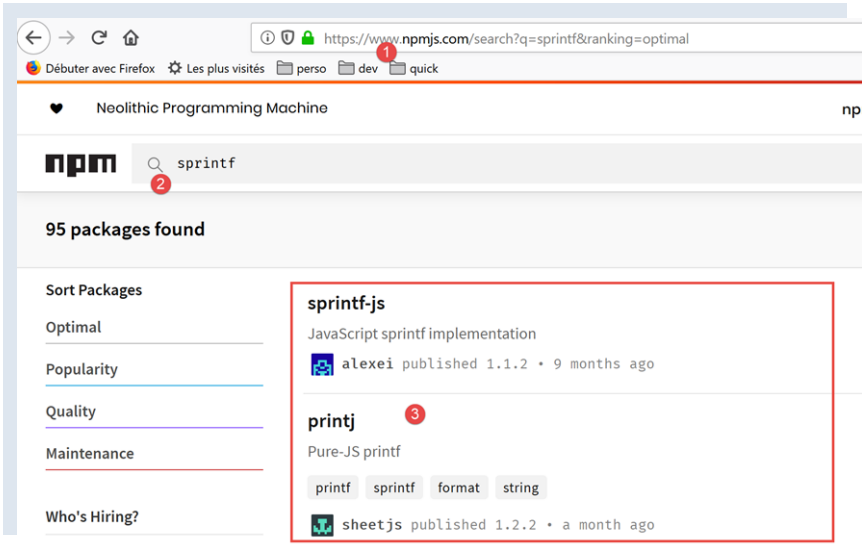
2.6.4 script [str-04]

La chaîne avec interpolation de variables reste insuffisante. Il n'est en effet pas possible de mettre une expression à la place de la variable dans l'expression ${variable}. Pour ceux qui ont programmé en C, il n'y a rien de tel que les fonctions **[printf, sprintf]** pour écrire ou construire des chaînes formatées. Des centaines de développeurs ont développé des milliers de packages Javascript formant un écosystème immense. Lorsqu'on a un besoin non satisfait nativement par Javascript, il est temps de chercher un package qui le satisfasse. Pour cela nous utilisons le gestionnaire de packages **[npm]**. Celui-ci dispose d'une option **[search]** qui permet de chercher une chaîne de caractères dans la description des packages. **[npm]** renvoie la liste des packages satisfaisant à la recherche. Nous allons donc chercher la chaîne **[sprintf]** dans la description des packages :



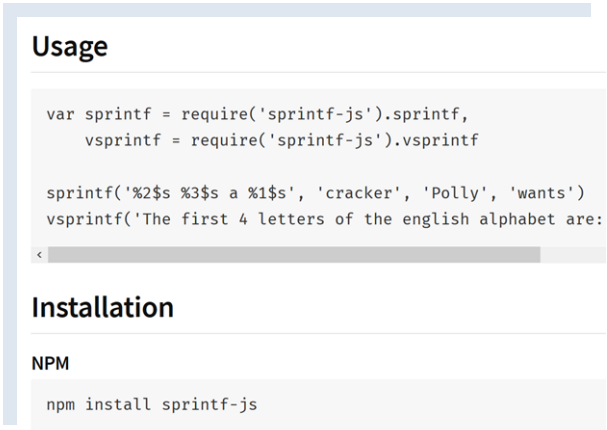
- dans la colonne **[4]**, les mots clés des packages de la colonne **[3]** ;
- dans la colonne **[5]**, la description des packages de la colonne **[3]** ;

L'étape suivante est d'aller sur le site de l'outil **[npm]**, **[https://www.npmjs.com/]** et de lire la description des packages :

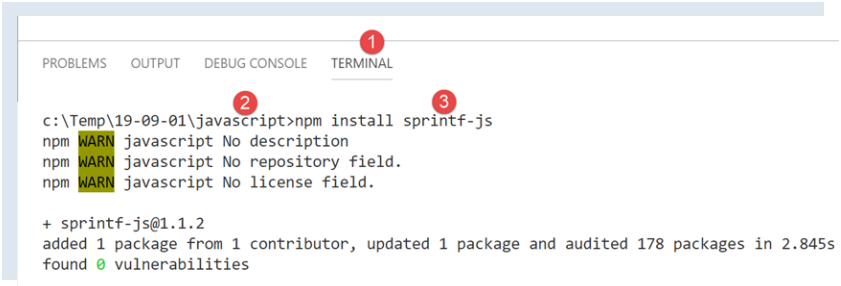


En **[3]**, on examine la liste des packages et on en choisit un.

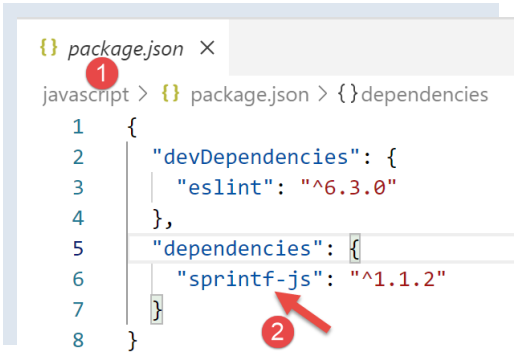
Dans la description du package, on trouve les informations pour l'installer et l'utiliser :



Nous installons le package **[sprintf-js]** dans un terminal de **[VSCode]** :



Cette installation va modifier le fichier **[package.json]** situé à la racine du dossier **[javascript] [2]** :



On voit ci-dessus que le package a été installé dans les **[dependencies]**, ç-à-d dans les packages nécessaires à l'exécution du projet. On rappelle qu'on met dans **[devDependencies]** les packages nécessaires uniquement pendant le développement du projet. Ils ne sont pas utilisés pendant l'exécution. Cette différence est importante lorsqu'il faut créer la version finale du projet pour sa mise en production. Il existe des outils pour :

- rassembler tous les fichiers jS nécessaires à l'exécution dans un seul fichier. Les packages des **[devDependencies]** ne sont donc pas inclus dans ce fichier final ;

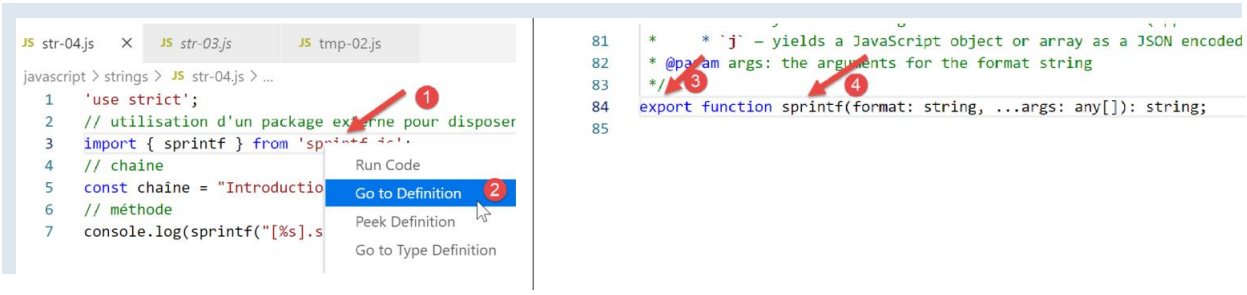
- minifier celui-ci, ç-à-d rendre sa taille la plus petite possible. Pour cela par exemple, tous les commentaires sont éliminés ;
- « obscurcir » le code pour le rendre difficilement compréhensible. Par exemple, les variables taux, salaire, impôt vont être remplacées par des variables a, b, c ;
- faire d'autres optimisations ;

Cette optimisation du fichier final d'un projet jS est utilisée en programmation web. Une application web peut dépendre de très nombreux fichiers Javascript. Le chargement de ceux-ci par un navigateur peut ralentir l'affichage de la première page de l'application. L'optimisation précédente vise à améliorer ce temps de chargement. Si le temps de chargement est jugé trop long par les utilisateurs, l'application ne sera pas utilisée.

Maintenant que nous disposons du package **[sprintf-js]**, il nous faut l'utiliser. C'est le script **[str-04]** :

```
1   'use strict';
2   // utilisation d'un package externe pour disposer de la fonction sprintf
3   import { sprintf } from 'sprintf-js';
4   // chaine
5   const chaîne = "Introduction à Javascript par l'exemple";
6   // méthode
7   console.log(sprintf("[%s].substr(3,2)=[%s]", chaîne, chaîne.substr(3, 2)));
```

Avec ECMAScript 6, on utilise le mot clé **[import]** pour **importer** un objet exporté par un package. Pour savoir ce qu'exporte le package, on peut aller voir son code :



- en **[1]**, clic droit sur le package importé ;
- en **[2]**, on veut en voir la définition ;
- en **[3-4]**, on voit que le package exporte une fonction appelée **[sprintf]** ;

La fonction **[sprintf]** du package **[sprintf-js]** est importée avec l'instruction :

```
import { sprintf } from 'sprintf-js';
```

Le code complet :

```
1   'use strict';
2   // utilisation d'un package externe pour disposer de la fonction sprintf
3   import { sprintf } from 'sprintf-js';
4   // chaine
5   const chaîne = "Introduction à Javascript par l'exemple";
6   // méthode
7   console.log(sprintf("[%s].substr(3,2)=[%s]", chaîne, chaîne.substr(3, 2)));
```

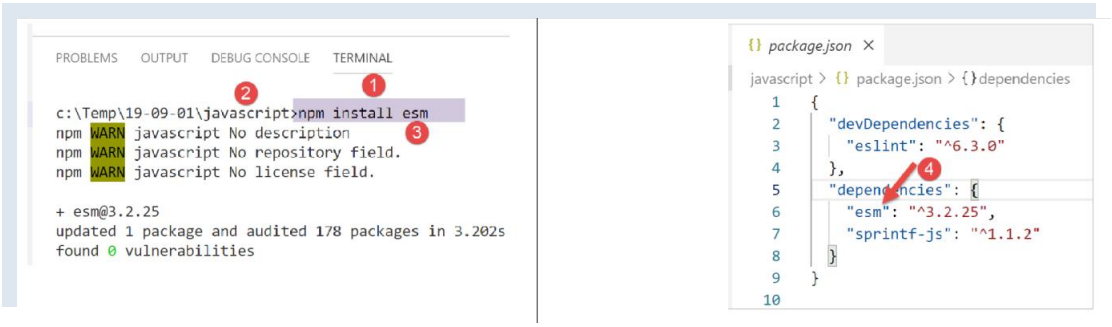
produit les résultats suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe "c:\Temp\19-09-01\javascript\strings\str-
    04.js"
2   c:\Temp\19-09-01\javascript\strings\str-04.js:3
3   import { sprintf } from 'sprintf-js';
4   ^
5
6   SyntaxError: Unexpected token {
7   at new Script (vm.js:79:7)
8   at createScript (vm.js:251:10)
9   at Object.runInThisContext (vm.js:303:10)
10  at Module._compile (internal/modules/cjs/loader.js:657:28)
```

Ligne 3, l'instruction **[import]** n'est pas comprise. Cela vient du fait que la version 10.15.1 de **[node.js]** utilisée dans ce cours (sept 2019) n'observe pas encore la norme ECMAScript pour l'importation de packages appelés modules. En 2019, **[node.js]** observe une norme de modules appelée CommonJS. L'intégration des modules ECMAScript par **[node.js]** est prévue en 2020. Là encore des

développeurs se sont mis à la tâche et ont produit des packages permettant l'utilisation de modules ES6 avec **[node.js]** dès maintenant (2019).

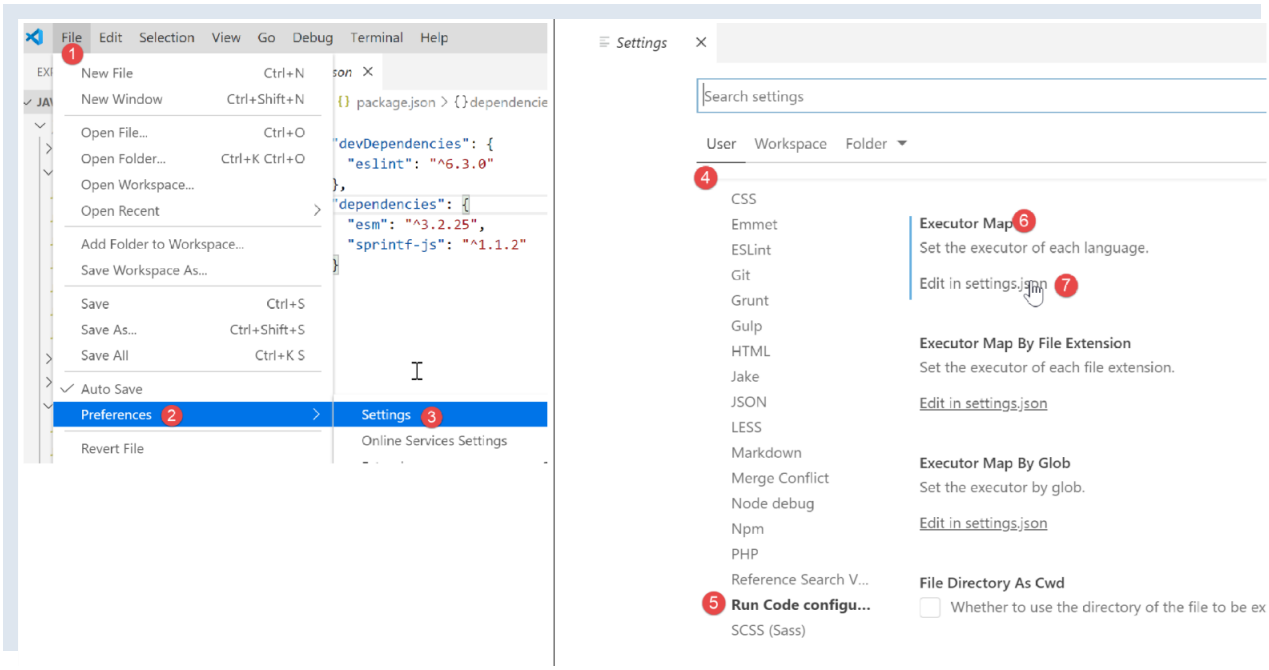
Nous allons utiliser un package appelé **[esm]** (**E**CMA**S**cript **M**odules). Nous l'installons dans un terminal du projet **[javascript]** :

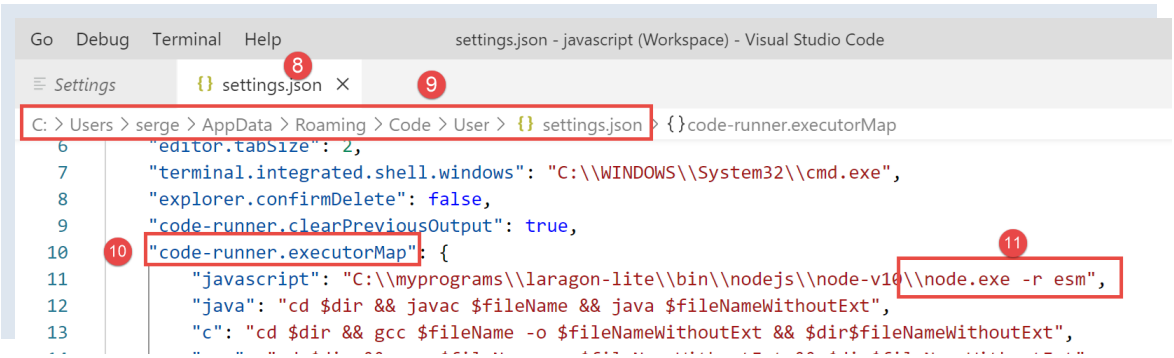


En **[4]**, on constate que l'installation du package **[esm]** **[1-3]** a modifié le fichier **[javascript/package.json]**.

Nous n'avons pas fini. Pour que le module **[esm]** soit utilisé par **[node.js]**, il faut lancer celui-ci avec l'argument **[-r esm]**.

Nous modifions dons la configuration de l'extension **[Code Runner]** de **[VSCode]** :

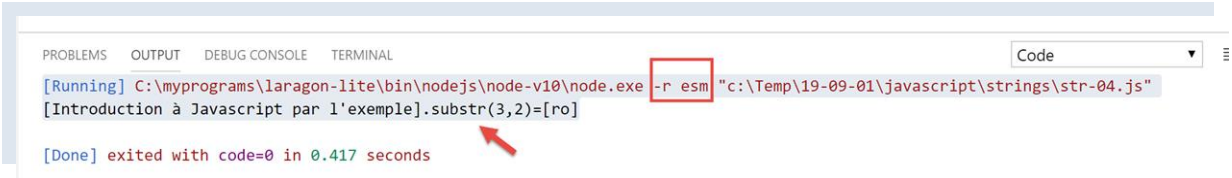




En **[11]**, on ajoute l'argument **[-r esm]** et on sauvegarde (Ctrl-S) la configuration.

Maintenant, nous pouvons exécuter le script **[str-04]** :

```
1   'use strict';
2   // utilisation d'un package externe pour disposer de la fonction sprintf
3   import { sprintf } from 'sprintf-js';
4   // chaine
5   const chaîne = "Introduction à Javascript par l'exemple";
6   // méthode substr
7   console.log(sprintf("[%s].substr(3,2)=[%s]", chaîne, chaîne.substr(3, 2)));
```



2.6.5 script `[str-05]`

Voici ce que dit la documentation sur la fonction `[sprintf]` :
The placeholders in the format string are marked by `%` and are followed by one or more of these elements, in this order:

- *An optional number followed by a `$` sign that selects which argument index to use for the value. If not specified, arguments will be placed in the same order as the placeholders in the input string.*
- *An optional `+` sign that forces to preceed the result with a plus or minus sign on numeric values. By default, only the `-` sign is used on negative numbers.*
- *An optional padding specifier that says what character to use for padding (if specified). Possible values are `0` or any other character precedeed by a `'` (single quote). The default is to pad with* spaces.
- *An optional `-` sign, that causes `sprintf` to left-align the result of this placeholder. The default is to right-align the result.*
- *An optional number, that says how many characters the result should have. If the value to be returned is shorter than this number, the result will be padded. When used with the `j` (JSON) type specifier, the padding length specifies the tab size used for indentation.*
- *An optional precision modifier, consisting of a `.` (dot) followed by a number, that says how many digits should be displayed for floating point numbers. When used with the `g` type specifier, it specifies the number of significant digits. When used on a string, it causes the result to be truncated.*
- *A type specifier that can be any of:*
- *`%` — yields a literal `%` character*
- *`b` — yields an integer as a binary number*
- *`c` — yields an integer as the character with that ASCII value*
- *`d` or `i` — yields an integer as a signed decimal number*
- *`e` — yields a float using scientific notation*
- *`u` — yields an integer as an unsigned decimal number*
- *`f` — yields a float as is; see notes on precision above*
- *`g` — yields a float as is; see notes on precision above*
- *`o` — yields an integer as an octal number*
- *`s` — yields a string as is*
- *`t` — yields `true` or `false`*
- *`T` — yields the type of the argument*[1]
- *`v` — yields the primitive value of the specified argument*
- *`x` — yields an integer as a hexadecimal number (lower-case)*
- *`X` — yields an integer as a hexadecimal number (upper-case)*
- *`j` — yields a JavaScript object or array as a JSON encoded string*

Le script **[script-05]** met en œuvre quelques-uns de ces formats :

```
1.  'use strict';
2.  // utilisation d'un package externe pour disposer de la fonction sprintf
3.  import { sprintf } from 'sprintf-js';
4.  // chaine
5.  const chaîne = "Javascript";
6.  // chaînes de caractères
7.  console.log(sprintf("[%s, %%s]=>[%s]", chaîne, chaîne));
8.  console.log(sprintf("[%s, %%20s]=>[%20s]", chaîne, chaîne));
9.  console.log(sprintf("[%s, %%-20s]=>[%-20s]", chaîne, chaîne));
10. // entiers
11. console.log(sprintf("[%d, %%d]=>[%d]", 10, 10));
12. console.log(sprintf("[%d, %%4d]=>[%4d]", 10, 10));
13. console.log(sprintf("[%d, %%-4d]=>[%-4d]", 10, 10));
14. console.log(sprintf("[%d, %%04d]=>[%04d]", 10, 10));
15. // réels
16. console.log(sprintf("[%f, %%f]=>[%f]", -10.5, -10.5));
17. console.log(sprintf("[%f, %%10.2f]=>[%10.2f]", -10.5, -10.5));
18. console.log(sprintf("[%f, %%-10.2f]=>[%-10.2f]", -10.5, -10.5));
19. console.log(sprintf("[%f, %%010.3f]=>[%010.3f]", -10.5, -10.5));
20. // json
21. console.log(sprintf("personne (%%j)=%j", { nom: "mathieu", âge: 34 }));
22. // type
23. console.log(sprintf("type personne (%%T)=%T", { nom: "mathieu", âge: 34 }));
24. // booléen
25. console.log(sprintf("booléen (%%t)=%t", 4 === 4));
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\strings\str-05.js"
2   [Javascript, %s]=>[Javascript]
3   [Javascript, %20s]=>[ Javascript]
4   [Javascript, %-20s]=>[Javascript ]
5   [10, %d]=>[10]
6   [10, %4d]=>[ 10]
7   [10, %-4d]=>[10 ]
8   [10, %04d]=>[0010]
9   [-10.5, %f]=>[-10.5]
10  [-10.5, %10.2f]=>[ -10.50]
11  [-10.5, %-10.2f]=>[-10.50 ]
12  [-10.5, %010.3f]=>[-00010.500]
13  personne (%j)={"nom":"mathieu","âge":34}
14  type personne (%T)=object
15  booléen (%t)=true
```

2.6.6 script [str-06]

Le script **[str-06]** présente quelques méthodes de la classe **[String]** utilisables également sur le type **[string]** :

```
1.  'use strict';
2.  // utilisation d'un package externe pour disposer de la fonction sprintf
3.  import { sprintf } from 'sprintf-js';
4.  // chaine
5.  const chaîne = "  Introduction à Javascript ";
6.  // quelques méthodes
7.  // substr(10,2) : 2 caractères à partir du n° 10
8.  console.log(sprintf("[%s].substr(10,2)=[%s]", chaîne, chaîne.substr(10, 2)));
9.  // trim : élimination des blancs de début et fin de chaîne (blanc=\b \t \r \n \f)
10. console.log(sprintf("[%s].trim()=[%s]", chaîne, chaîne.trim()));
11. // toLowerCase : transformation en minuscules
12. console.log(sprintf("[%s].toLowerCase=[%s]", chaîne, chaîne.toLowerCase()));
13. // toUpperCase : transformation en majuscules
14. console.log(sprintf("[%s].toUpperCase=[%s]", chaîne, chaîne.toUpperCase()));
15. // indexOf : position d'une chaîne  cherchée dans la chaîne, -1 si la sous-chaîne n'existe pas
16. console.log(sprintf("[%s].indexOf('Java')=[%s]", chaîne, chaîne.indexOf('Java')));
17. console.log(sprintf("[%s].trim().indexOf('abcd')=[%s]", chaîne, chaîne.indexOf('abcd')));
18. // includes : vrai si la chaîne cherchée est dans la chaîne
19. console.log(sprintf("[%s].includes('Java')=[%s]", chaîne, chaîne.includes('Java')));
20. // length : longueur de la chaîne - n'est pas une méthode mais une propriété
21. console.log(sprintf("[%s].length=[%s]", chaîne, chaîne.length));
22. // slice (7,10) : chaînes des caractères n° 7 à 9
23. console.log(sprintf("[%s].slice(7,10)=[%s]", chaîne, chaîne.slice(7, 10)));
24. // match : cherche une expression dans la chaîne - cette expression peut être une expression régulière
```

```
25. // /intro/i : expression régulière désignant la chaîne [intro] en majuscules ou minuscules
26. // rend la chaîne trouvée
27. console.log(sprintf("[%s].match(/intro/i)=[%s]", chaîne, chaîne.match(/intro/i)));
28. // replace : remplace chaine1 par chaine2 dans chaîne
29. // remplace la 1ère occurrence de i par x
30. console.log(sprintf("[%s].replace('i','x')=[%s]", chaîne, chaîne.replace('i', 'x')));
31. // remplace toutes les occurrences de i par x
32. // /i/g est une expression régulière désignant toutes (g) les occurrences de i
33. console.log(sprintf("[%s].replace(/i/g,'x')=[%s]", chaîne, chaîne.replace(/i/g, 'x')));
34. // split : divise la chaîne en mots séparés par le paramètre de split
35. // rend le tableau de ces mots
36. // /\s*/ : mots séparés par 0 ou plusieurs espaces
37. console.log(sprintf("[%s].split(/\\s*/)=[%s]", chaîne, chaîne.split(/\s*/)));
38. // /\s+/ : mots séparés par un ou plusieurs espaces
39. console.log(sprintf("[%s].split(/\\s+/)=[%s]", chaîne, chaîne.split(/\s+/)));
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\strings\str-06.js"
2   [ Introduction à Javascript ].substr(10,2)=[ti]
3   [ Introduction à Javascript ].trim()=[Introduction à Javascript]
4   [ Introduction à Javascript ].toLowerCase=[ introduction à javascript ]
5   [ Introduction à Javascript ].toUpperCase=[ INTRODUCTION À JAVASCRIPT ]
6   [ Introduction à Javascript ].indexOf('Java')=[17]
7   [ Introduction à Javascript ].trim().indexOf('abcd')=[-1]
8   [ Introduction à Javascript ].includes('Java')=[true]
9   [ Introduction à Javascript ].length=[28]
10  [ Introduction à Javascript ].slice(7,10)=[duc]
11  [ Introduction à Javascript ].match(/intro/i)=[Intro]
12  [ Introduction à Javascript ].replace('i','x')=[ Introductxon à Javascript ]
13  [ Introduction à Javascript ].replace(/i/g,'x')=[ Introductxon à Javascrxpt ]
14  [ Introduction à Javascript ].split(/\s*/)=[,I,n,t,r,o,d,u,c,t,i,o,n,à,J,a,v,a,s,c,r,i,p,t,]
15  [ Introduction à Javascript ].split(/\s+/)=[,Introduction,à,Javascript,]
```

2.7 Expressions régulières

2.7.1 script [regexp-01]

Dans le cours PHP, nous avions utilisé le code suivant pour illustrer les expressions régulières de PHP 7 :

```
1   <?php
2
3   // type strict pour les paramètres de fonctions
4   declare (strict_types=1);
5
6   // expressions régulières en php
7   // récupérer les différents champs d'une chaîne
8   // le modèle : une suite de chiffres entourée de caractères quelconques
9   // on ne veut récupérer que la suite de chiffres
10  $modèle = "/(\d+)/";
11  // on confronte la chaîne au modèle
12  compareModele2Chaine($modèle, "xyz1234abcd");
13  compareModele2Chaine($modèle, "12 34");
14  compareModele2Chaine($modèle, "abcd");
15
16  // le modèle : une suite de chiffres entourée de caractères quelconques
17  // on veut la suite de chiffres ainsi que les champs qui suivent et précèdent
18  $modèle = "/^(.*?)(\d+)(.*?)$/";
19  // on confronte la chaîne au modèle
20  compareModele2Chaine($modèle, "xyz1234abcd");
21  compareModele2Chaine($modèle, "12 34");
22  compareModele2Chaine($modèle, "abcd");
23
24  // le modèle - une date au format jj/mm/aa
25  $modèle = "/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/";
26  compareModele2Chaine($modèle, "10/05/97");
```

```
27  compareModele2Chaine($modèle, "  04/04/01  ");
28  compareModele2Chaine($modèle, "5/1/01");
29
30  // le modèle - un nombre décimal
31  $modèle = "/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/";
32  compareModele2Chaine($modèle, "187.8");
33  compareModele2Chaine($modèle, "-0.6");
34  compareModele2Chaine($modèle, "4");
35  compareModele2Chaine($modèle, ".6");
36  compareModele2Chaine($modèle, "4.");
37  compareModele2Chaine($modèle, " + 4");
38
39  // fin
40  exit;
41
42  // ------------------------------------------------------------------------
43  function compareModele2Chaine(string $modèle, string $chaîne): void {
44    // compare la chaîne $chaîne au modèle $modèle
45    // on confronte la chaîne au modèle
46    $champs = [];
47    $correspond = preg_match($modèle, $chaîne, $champs);
48    // affichage résultats
49    print "\nRésultats($modèle,$chaîne)\n";
50    if ($correspond) {
51      for ($i = 0; $i < count($champs); $i++) {
52        print "champs[$i]=$champs[$i]\n";
53      }
54    } else {
55      print "La chaîne [$chaîne] ne correspond pas au modèle [$modèle]\n";
56    }
57  }
```

Nous transposons ce code en Javascript de la façon suivante :

```
1.  'use strict';
2.
3.  /// expressions régulières en javascript
4.  // récupérer les différents champs d'une chaîne
5.  // le modèle : une suite de chiffres entourée de caractères quelconques
6.  // on ne veut récupérer que la suite de chiffres
7.  let modèle = /(\d+)/;
8.  console.log("type d'une expression régulière : ", typeof (modèle));
9.  // on confronte la chaîne au modèle
10. compareModèleToChaîne(modèle, "xyz1234abcd");
11. compareModèleToChaîne(modèle, "12 34");
12. compareModèleToChaîne(modèle, "abcd");
13.
14. // le modèle : une suite de chiffres entourée de caractères quelconques
15. // on veut la suite de chiffres ainsi que les champs qui suivent et précèdent
16. modèle = /^(.*?)(\d+)(.*?)$/;
17. // on confronte la chaîne au modèle
18. compareModèleToChaîne(modèle, "xyz1234abcd");
19. compareModèleToChaîne(modèle, "12 34");
20. compareModèleToChaîne(modèle, "abcd");
21.
22. // le modèle - une date au format jj/mm/aa
23. modèle = /^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/;
24. compareModèleToChaîne(modèle, "10/05/97");
25. compareModèleToChaîne(modèle, "  04/04/01  ");
26. compareModèleToChaîne(modèle, "5/1/01");
27.
28. // le modèle - un nombre décimal
29. modèle = /^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/;
30. compareModèleToChaîne(modèle, "187.8");
31. compareModèleToChaîne(modèle, "-0.6");
32. compareModèleToChaîne(modèle, "4");
33. compareModèleToChaîne(modèle, ".6");
34. compareModèleToChaîne(modèle, "4.");
35. compareModèleToChaîne(modèle, " + 4");
36.
37. // ------------------------------------------------------------------------
38. function compareModèleToChaîne(modèle, chaîne) {
39.   // compare la chaîne [chaîne] au modèle [modèle]
40.   console.log(`---------- chaîne=${chaîne}, modèle=${modèle}`)
41.   // on confronte la chaîne au modèle
42.   const result1 = modèle.exec(chaîne);
43.   console.log(`comparaison avec exec=`, result1);
```

```
44.   // une autre façon de faire
45.   const result2 = chaîne.match(modèle);
46.   console.log(`comparaison avec match=`, result2);
47. }
```

Commentaires

- les codes PHP et Javascript sont très proches l'un de l'autre ;
- ligne 7 : on notera qu'en Javascript l'expression régulière n'est pas une chaîne de caractères mais un objet. On ne met pas de guillemets ou d'apostrophes autour de l'expression ;
- lignes 42 et 45 : il y a deux méthodes pour obtenir le même résultat ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\regexp\regexp-01.js"
2   type d'une expression régulière : object
3   ---------- chaîne=xyz1234abcd, modèle=/(\d+)/
4   comparaison avec exec= [ '1234',
5   '1234',
6   index: 3,
7   input: 'xyz1234abcd',
8   groups: undefined ]
9   comparaison avec match= [ '1234',
10  '1234',
11  index: 3,
12  input: 'xyz1234abcd',
13  groups: undefined ]
14  ---------- chaîne=12 34, modèle=/(\d+)/
15  comparaison avec exec= [ '12', '12', index: 0, input: '12 34', groups: undefined ]
16  comparaison avec match= [ '12', '12', index: 0, input: '12 34', groups: undefined ]
17  ---------- chaîne=abcd, modèle=/(\d+)/
18  comparaison avec exec= null
19  comparaison avec match= null
20  ---------- chaîne=xyz1234abcd, modèle=/^(.*?)(\d+)(.*?)$/
21  comparaison avec exec= [ 'xyz1234abcd',
22  'xyz',
23  '1234',
24  'abcd',
25  index: 0,
26  input: 'xyz1234abcd',
27  groups: undefined ]
28  comparaison avec match= [ 'xyz1234abcd',
29  'xyz',
30  '1234',
31  'abcd',
32  index: 0,
33  input: 'xyz1234abcd',
34  groups: undefined ]
35  ---------- chaîne=12 34, modèle=/^(.*?)(\d+)(.*?)$/
36  comparaison avec exec= [ '12 34',
37  '',
38  '12',
39  ' 34',
40  index: 0,
41  input: '12 34',
42  groups: undefined ]
43  comparaison avec match= [ '12 34',
44  '',
45  '12',
46  ' 34',
47  index: 0,
48  input: '12 34',
49  groups: undefined ]
50  ---------- chaîne=abcd, modèle=/^(.*?)(\d+)(.*?)$/
51  comparaison avec exec= null
52  comparaison avec match= null
53  ---------- chaîne=10/05/97, modèle=/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/
54  comparaison avec exec= [ '10/05/97',
55  '10',
56  '05',
57  '97',
58  index: 0,
59  input: '10/05/97',
60  groups: undefined ]
```

```
comparaison avec match= [ '10/05/97',
'10',
'05',
'97',
index: 0,
input: '10/05/97',
groups: undefined ]
---------- chaîne= 04/04/01 , modèle=/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/
comparaison avec exec= [ ' 04/04/01 ',
'04',
'04',
'01',
index: 0,
input: ' 04/04/01 ',
groups: undefined ]
comparaison avec match= [ ' 04/04/01 ',
'04',
'04',
'01',
index: 0,
input: ' 04/04/01 ',
groups: undefined ]
---------- chaîne=5/1/01, modèle=/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/
comparaison avec exec= null
comparaison avec match= null
---------- chaîne=187.8, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '187.8',
'',
'187.8',
index: 0,
input: '187.8',
groups: undefined ]
comparaison avec match= [ '187.8',
'',
'187.8',
index: 0,
input: '187.8',
groups: undefined ]
---------- chaîne=-0.6, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '-0.6', '-', '0.6', index: 0, input: '-0.6', groups: undefined ]
comparaison avec match= [ '-0.6', '-', '0.6', index: 0, input: '-0.6', groups: undefined ]
---------- chaîne=4, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '4', '', '4', index: 0, input: '4', groups: undefined ]
comparaison avec match= [ '4', '', '4', index: 0, input: '4', groups: undefined ]
---------- chaîne=.6, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '.6', '', '.6', index: 0, input: '.6', groups: undefined ]
comparaison avec match= [ '.6', '', '.6', index: 0, input: '.6', groups: undefined ]
---------- chaîne=4., modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ '4.', '', '4.', index: 0, input: '4.', groups: undefined ]
comparaison avec match= [ '4.', '', '4.', index: 0, input: '4.', groups: undefined ]
---------- chaîne= + 4, modèle=/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/
comparaison avec exec= [ ' + 4', '+', '4', index: 0, input: ' + 4', groups: undefined ]
comparaison avec match= [ ' + 4', '+', '4', index: 0, input: ' + 4', groups: undefined ]
```

Les méthodes **[regexp.exec]** et **[string.match]** donnent les mêmes résultats :

- **[null]** s'il n'y a pas de correspondances entre la chaîne et son modèle ;
- un tableau t, s'il y a correspondance avec :
  - t**[0]** : la chaîne correspondant au modèle ;
  - t**[1]** : la chaîne correspondant à la 1ère parenthèse du modèle ;
  - t**[2]** : la chaîne correspondant à la 2ième parenthèse du modèle ;
  - …
  - t**[input]** : la chaîne entière dans laquelle on a cherché le modèle ;

2.7.2 script [regexp-02]

Parfois on ne souhaite pas récupérer des éléments de la chaîne testée mais seulement savoir si elle correspond au modèle :

```
1.  'use strict';
2.
3.  /// expressions régulières en javascript
4.  // récupérer les différents champs d'une chaîne
5.  // le modèle : une suite de chiffres entourée de caractères quelconques
6.  // on ne veut récupérer que la suite de chiffres
```

```
7.  let modèle = /\d+/;
8.  console.log("type d'une expression régulière : ", typeof (modèle));
9.  // on confronte la chaîne au modèle
10. compareModèleToChaîne(modèle, "xyz1234abcd");
11. compareModèleToChaîne(modèle, "12 34");
12. compareModèleToChaîne(modèle, "abcd");
13.
14. // le modèle : une suite de chiffres entourée de caractères quelconques
15. // on veut la suite de chiffres ainsi que les champs qui suivent et précèdent
16. modèle = /^.*?\d+.*?$/;
17. // on confronte la chaîne au modèle
18. compareModèleToChaîne(modèle, "xyz1234abcd");
19. compareModèleToChaîne(modèle, "12 34");
20. compareModèleToChaîne(modèle, "abcd");
21.
22. // le modèle - une date au format jj/mm/aa
23. modèle = /^\s*\d\d\/\d\d\/\d\d\s*$/;
24. compareModèleToChaîne(modèle, "10/05/97");
25. compareModèleToChaîne(modèle, "  04/04/01  ");
26. compareModèleToChaîne(modèle, "5/1/01");
27.
28. // le modèle - un nombre décimal
29. modèle = /^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/;
30. compareModèleToChaîne(modèle, "187.8");
31. compareModèleToChaîne(modèle, "-0.6");
32. compareModèleToChaîne(modèle, "4");
33. compareModèleToChaîne(modèle, ".6");
34. compareModèleToChaîne(modèle, "4.");
35. compareModèleToChaîne(modèle, " + 4");
36.
37. // ----------------------------------------------------------------------
38. function compareModèleToChaîne(modèle, chaîne) {
39.   // test
40.   const correspond = modèle.test(chaîne);
41.   // compare la chaîne [chaîne] au modèle [modèle]
42.   console.log(`----------- chaîne=${chaîne}, modèle=${modèle}, correspond=${correspond}`);
43. }
```

Commentaires

- **[regexp-02]** reprend le code de **[regexp-01]** avec les différences suivantes :
  - on ne souhaite pas récupérer des éléments de la chaîne testée. Aussi a-t-on enlevé les parenthèses dans les expression régulières utilisées ;
  - ligne 40 : on utilise la méthode **[Regexp.test]** pour savoir si une chaîne de caractères vérifie une expression régulière ;

Les résultats de l'exécution sont les suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\cours\regexp\regexp-02.js"
2   type d'une expression régulière :  object
3   ----------- chaîne=xyz1234abcd, modèle=/\d+/, correspond=true
4   ----------- chaîne=12 34, modèle=/\d+/, correspond=true
5   ----------- chaîne=abcd, modèle=/\d+/, correspond=false
6   ----------- chaîne=xyz1234abcd, modèle=/^.*?\d+.*?$/, correspond=true
7   ----------- chaîne=12 34, modèle=/^.*?\d+.*?$/, correspond=true
8   ----------- chaîne=abcd, modèle=/^.*?\d+.*?$/, correspond=false
9   ----------- chaîne=10/05/97, modèle=/^\s*\d\d\/\d\d\/\d\d\s*$/, correspond=true
10  ----------- chaîne=  04/04/01  , modèle=/^\s*\d\d\/\d\d\/\d\d\s*$/, correspond=true
11  ----------- chaîne=5/1/01, modèle=/^\s*\d\d\/\d\d\/\d\d\s*$/, correspond=false
12  ----------- chaîne=187.8, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
13  ----------- chaîne=-0.6, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
14  ----------- chaîne=4, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
15  ----------- chaîne=.6, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
16  ----------- chaîne=4., modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
17  ----------- chaîne= + 4, modèle=/^\s*[+|-]?\s*\d+\.\d*|\.\d+|\d+\s*$/, correspond=true
18
19  [Done] exited with code=0 in 0.269 seconds
```

2.8 Les fonctions

2.8.1 script [func-01]

Le script s'intéresse au mode de passage des paramètres d'une fonction :

- passage par valeur pour nombres, chaînes et booléens ;
- passage par référence pour les tableaux, objets littéraux et fonctions ;

```
1.  'use strict';
2.  // mode de passage des paramètres d'une fonction
3.  // ----------------------nombre - passage par valeur
4.  function doSomethingWithNumber(param) {
5.    param++;
6.    console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par référence]=", param === c
    ount);
7.  }
8.  // code d'appel
9.  let count = 10;
10. doSomethingWithNumber(count);
11. console.log("[count outside function]=", count);
12.
13. // ---------------------- chaîne - passage par valeur
14. function doSomethingWithString(param) {
15.   param += " xyz"
16.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par référence]=", param === t
    ext);
17. }
18. // code d'appel
19. let text = "abcd";
20. doSomethingWithString(text);
21. console.log("[text outside function]=", text);
22.
23. // ---------------------- booléen - passage par valeur
24. function doSomethingWithBoolean(param) {
25.   param = !param;
26.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par référence]=", param === b
    ool);
27. }
28. // code d'appel
29. let bool = true;
30. doSomethingWithBoolean(bool);
31. console.log("bool [outside function]=", bool);
32.
33. // ---------------------- tableau - passage par référence
34. function doSomethingWithArray(param) {
35.   param.push(1000);
36.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par référence]=", param === t
    ab);
37. }
38. // code d'appel
39. const tab = [10, 20, 30];
40. doSomethingWithArray(tab);
41. console.log("[tab outside function]=", tab);
42.
43. // ---------------------- objet - passage par référence
44. function doSomethingWithObject(param) {
45.   param.unePropriétéNouvelle = "xyz";
46.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par référence]=", param === o
    bj);
47. }
48. // code d'appel
49. const obj = [10, 20, 30];
50. doSomethingWithObject(obj);
51. console.log("[obj outside function]=", obj);
52.
53. // ---------------------- fonction - passage par référence
54. function doSomethingWithFunction(param) {
55.   // une chose plutôt bizarre qui marche pourtant
```

```
56.   param.unePropriétéNouvelle = "xyz";
57.   console.log("[param inside function]=", param, "[type]=", typeof (param), "[passage par référence]=", param === f
    );
58. }
59. // code d'appel
60. const f = x => x + 4;
61. doSomethingWithFunction(f);
62. console.log("[f outside function]=", f, f.unePropriétéNouvelle, typeof (f));
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\fonctions\func-01.js"
2   [param inside function]= 11 [type]= number [passage par référence]= false
3   [count outside function]= 10
4   [param inside function]= abcd xyz [type]= string [passage par référence]= false
5   [text outside function]= abcd
6   [param inside function]= false [type]= boolean [passage par référence]= false
7   bool [outside function]= true
8   [param inside function]= [ 10, 20, 30, 1000 ] [type]= object [passage par référence]= true
9   [tab outside function]= [ 10, 20, 30, 1000 ]
10  [param inside function]= [ 10, 20, 30, 'unePropriétéNouvelle': 'xyz' ] [type]= object [passage par
    référence]= true
11  [obj outside function]= [ 10, 20, 30, 'unePropriétéNouvelle': 'xyz' ]
12  [param inside function]= x => x + 4 [type]= function [passage par référence]= true
13  [f outside function]= x => x + 4 xyz function
```

2.8.2 script [func-02]

Le script suivant montre que le type **[function]** est un type de donnée comme un autre et qu'une variable peut avoir ce type. Il montre également deux façons de définir une fonction :

- l'une avec le mot clé **[function]** ;
- l'autre avec la notation « flèche » => ;

```
1.  'use strict';
2.  // on peut affecter une fonction à une variable
3.  const variable1 = function (a, b) {
4.    return a + b;
5.  };
6.  console.log("typeof(variable1)=", typeof (variable1));
7.  // la variable peut ensuite s'utiliser comme une fonction
8.  console.log("variable1(10,12)=", variable1(10, 12));
9.  // la définition de la fonction peut se faire avec la notation =>
10. const variable2 = (a, b, c) => {
11.   return a - b + c;
12. };
13. console.log("variable2(10,12,14)=", variable2(10, 12, 14));
14. // on peut ne pas mettre les accolades s'il n'y a qu'une expression dans le code de la fonction
15. // cette expression est alors la valeur de retour de la fonction
16. const variable3 = (a, b, c) => a + b + c;
17. console.log("variable3(10,12,14)=", variable3(10, 12, 14));
```

Exécution

```
1.  [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\fonctions\func-02.js"
2.  typeof(variable1)= function
3.  variable1(10,12)= 22
4.  variable2(10,12,14)= 12
5.  variable3(10,12,14)= 36
```

2.8.3 script [func-03]

Ce script revient sur la possibilté de passer une fonction en paramètre à une autre fonction. Ce procédé est abondamment utilisé dans les frameworks Javascript.

```
1.  'use strict';
2.  // les paramètres d'une fonction peuvent être de type [fonction]
3.
4.  // fonction f1
```

```
5.  function f1(param1, param2) {
6.    return param1 + param2 + 10;
7.  }
8.  // fonction f2
9.  function f2(param1, param2) {
10.   return param1 + param2 + 20;
11. }
12. // fonction g avec une fonction f en paramètre
13. function g(param1, param2, f) {
14.   return f(param1, param2) + 100;
15. }
16. // utilisations de g
17. console.log(g(0, 10, f1));
18. console.log(g(0, 10, f2));
19. // le paramètre effectif de type fonction peut être passé en direct - forme 1
20. console.log(g(0, 10, (param1, param2) => {
21.   return param1 + param2 + 30;
22. }));
23. // le paramètre effectif de type fonction peut être passé en direct - forme 2
24. console.log(g(0, 10, function (param1, param2) {
25.   return param1 + param2 + 40;
26. }));
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\fonctions\func-03.js"
2   120
3   130
4   140
5   150
```

2.8.4 script `[func-04]`

Le script suivant montre qu'une fonction Javascript peut se comporter comme une classe :

```
1.  'use strict';
2.  // une fonction peut être utilisée comme un objet
3.
4.  // une coquille vide
5.  function f() {
6.
7.  }
8.  // à qui on attribue des propriétés de l'extérieur
9.  f.prop1 = "val1";
10. f.show = function () {
11.   console.log(this.prop1);
12. }
13. // utilisation de f
14. f.show();
15.
16. // une fonction g fonctionnant comme une classe
17. function g() {
18.   this.prop2 = "val2";
19.   this.show = function () {
20.     console.log(this.prop2);
21.   }
22. }
23. // instanciation de la fonction avec [new] (à cause du mot clé this)
24. new g().show();
```

Commentaires

- lignes 5-7 : le corps de la fonction f ne définit aucune propriété ;
- lignes 9-12 : on donne de l'extérieur des propriétés à la fonction f ;
- ligne 14 : utilisation de la function (objet) f. Notez qu'on n'écrit pas **`[f()]`** mais simplement **`[f]`**. On a là la notation d'un objet ;
- lignes 17-22 : on définit une fonction **`[g]`** comme si c'était une classe avec propriétés et méthodes ;
- ligne 24 : la fonction **`[g]`** est instanciée par **`[new g()]`** ;

Résultats de l'exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\classes\class-00.js"
```

```
2   val1
3   val2
```

ES6 a introduit la notion de classe qui nous permet désormais d'éviter de passer par des fonctions pour avoir des classes.

2.8.5 script [func-05]

Le script **[func-05]** montre l'usage d'un opérateur appelé **[rest operator]** :

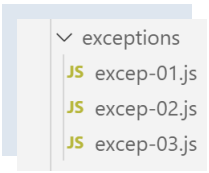
```
1.  'use strict';
2.  // rest operator
3.  function f(arg1, ...otherArgs) {
4.    // 1er argument
5.    console.log("arg1=", arg1);
6.    // les autres arguments
7.    let i = 0;
8.    otherArgs.forEach(element => {
9.      console.log("otherArguments[", i, "]=", element);
10.     i++;
11.   });
12. }
13.
14. // appel
15. f(1, "deux", "trois", { x: 2, y: 3 })
```

- ligne 3 : la notation **[...otherArgs]** fait qu'avec un appel de type f(param1, param2, param3) on aura ligne 3 :
  - arg1=param1
  - otherArgs=**[param2, param3]**. **[otherArgs]** est donc un tableau qui rassemble tous les paramètres effectifs passés derrière **[param1]** ;

Les résultats de l'application sont les suivants :

```
1   arg1= 1
2   otherArguments[ 0 ]= deux
3   otherArguments[ 1 ]= trois
4   otherArguments[ 2 ]= { x: 2, y: 3 }
```

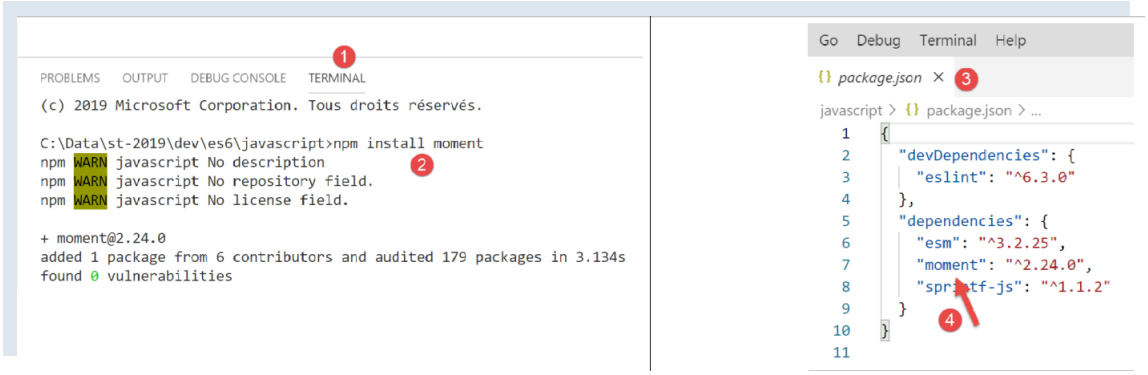
2.9 Les erreurs et exceptions



Javascript n'a pas un système d'exceptions très évolué. Il offre néanmoins l'instruction **[throw]** qui permet de signaler une erreur ainsi que la structure try / catch / finally qui permet d'intercepter ces erreurs.

2.9.1 script [excep-01]

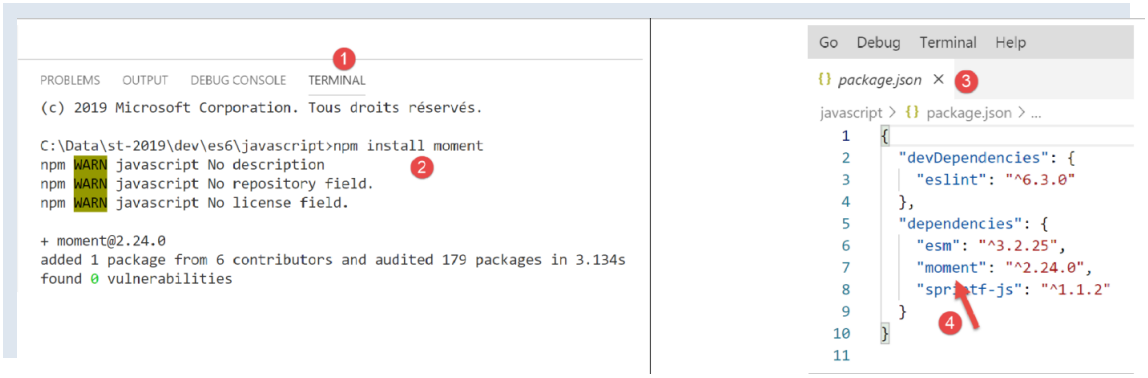
Dans le script qui suit, nous allons afficher la date du moment sous la forme **[heures:minutes:secondes:millisecondes]**. Pour ce faire, nous allons utiliser une bibliothèques jS appelée **[moment.js]**. Nous l'installons, comme d'habitude, avec l'outil **[npm]** :



Le code du script est le suivant :

```
1   'use strict';
2
3   // package moment
4   import moment from 'moment';
5
6   ...
```

Pour savoir comment écrire la ligne **[import]** de la ligne 4, on peut regarder la définition du module **[moment]** :



- en **[1-2]**, on va à la définition du module ;
- en **[3]**, on a un fichier Typescript, pas Javascript. A l'exécution, ce fichier Typescript est compilé en fichier Javascript avant d'être utilisé ;
- en **[4]**, on cherche les instructions **[export]** (Ctrl-F) ;
- en **[5]**, l'instruction exporte l'objet **[moment]**. Celui-ci peut être importé en ES6 de la façon suivante :

```
import moment from 'moment';
```

On peut utiliser n'importe quel nom pour importer l'objet **[moment]**, par exemple :

```
import m from 'moment';
```

Revenons au code du script :

```
1.
2.  'use strict';
3.
4.  // package moment
5.  import moment from 'moment';
6.
7.  // principe du try / catch / finally
8.  for (let i = 0; i < 10; i++) {
9.    // date - heure du moment courant
10.   const now = Date.now();
11.   // formatage heure pour avoir les millisecondes
12.   const time = moment(now).format("HH:mm:ss:SSS");
13.   // les millisecondes
14.   const milli = Number(time.substr(time.length - 3));
```

```
15.   // affichage
16.   console.log("--------------------itération n° ", i, "à", time);
17.   try {
18.     // nbre aléatoire
19.     const nbre = milli % 2;
20.     if (nbre === 0) {
21.       // lancer un msg d'erreur
22.       throw "erreur";
23.     }
24.     // si on arrive ici c'est qu'il n'y a pas eu d'erreur
25.     console.log("pas d'erreur");
26.   } catch (error) {
27.     // si on arrive ici, c'est qu'il y a eu erreur
28.     console.log("erreur1=", error);
29.   } finally {
30.     // exécuté dans tous les cas erreur ou pas
31.     console.log("finally")
32.   }
33. }
```

Commentaires

- ligne 4 : import de la bibliothèque **[moment]** ;
- le principe du script est de boucler 10 fois (ligne 7). A chaque tour de boucle, on récupère l'heure du moment sous la forme **[heures:minutes:secondes:millisecondes]** (lignes 8-13) ;
- si le nombre de millisecondes est pair, on lance un message d'erreur (lignes 19-22) ;
- il s'agit ici de comprendre le fonctionnement du try / catch / finally

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\exceptions\excep-01.js"
2   --------------------itération n° 0 à 17:57:04:440
3   erreur1= erreur
4   finally
5   --------------------itération n° 1 à 17:57:04:447
6   pas d'erreur
7   finally
8   --------------------itération n° 2 à 17:57:04:448
9   erreur1= erreur
10  finally
11  --------------------itération n° 3 à 17:57:04:448
12  erreur1= erreur
13  finally
14  --------------------itération n° 4 à 17:57:04:448
15  erreur1= erreur
16  finally
17  --------------------itération n° 5 à 17:57:04:448
18  erreur1= erreur
19  finally
20  --------------------itération n° 6 à 17:57:04:448
21  erreur1= erreur
22  finally
23  --------------------itération n° 7 à 17:57:04:448
24  erreur1= erreur
25  finally
26  --------------------itération n° 8 à 17:57:04:449
27  pas d'erreur
28  finally
29  --------------------itération n° 9 à 17:57:04:449
30  pas d'erreur
31  finally
```

On voit que la clause **[finally]** est toujours exécutée qu'il y ait erreur ou pas.

2.9.2 script [excep-02]

Ce script que l'instruction **[throw]** peut lancer n'importe quel type de donnée et que cette donnée est récupérée intégralement par la clause **[catch]**.

```
1.  'use strict';
2.
3.  // on peut "lancer" (throw) à peu près n'importe quoi pour signaler une erreur
4.  let i = 0;
```

```
5.  console.log("--------------------essai n° ", i);
6.  // lancer une chaîne de caractères
7.  try {
8.    throw "msg d'erreur";
9.  } catch (error) {
10.   // il y a eu erreur
11.   console.log("erreur=[", error, "], type=", typeof (error));
12. }
13. // lancer un objet littéral
14. i++;
15. console.log("--------------------essai n° ", i);
16. try {
17.   throw [1, 2, 3]
18. } catch (error) {
19.   // il y a eu erreur
20.   console.log("erreur=[", error, "], type=", typeof (error));
21. }
22. // lancer un objet
23. i++;
24. console.log("--------------------essai n° ", i);
25. try {
26.   throw { nom: "hercule", pays: "grèce antique" }
27. } catch (error) {
28.   // il y a eu erreur
29.   console.log("erreur=[", error, "], type=", typeof (error));
30. }
31. // lancer un type Error
32. i++;
33. console.log("--------------------essai n° ", i);
34. try {
35.   throw new Error("erreur de connexion au réseau");
36. } catch (error) {
37.   // il y a eu erreur
38.   console.log("erreur=[", error, "], type=", typeof (error));
39. }
40. // lancer un type Error
41. i++;
42. console.log("--------------------essai n° ", i);
43. try {
44.   throw new Error("erreur de connexion au réseau");
45. } catch (error) {
46.   // il y a eu erreur - le message est dans [error.message]
47.   console.log("erreur.message=[", error.message, "], type(error)=", typeof (error));
48. }
```

Commentaires

- lignes 35, 44 : **[Error]** est une classe Javascript dont le constructeur admet comme 1er paramètre facultatif un message d'erreur. Ce message peut être récupéré dans la propriété **[Error.message]** (ligne 47) ;
- il existe d'autres classes que **[Error]** pour signaler une erreur : **[EvalError, InternalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError) ;**

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\exceptions\excep-02.js"
2   --------------------essai n° 0
3   erreur=[ msg d'erreur ], type= string
4   --------------------essai n° 1
5   erreur=[ [ 1, 2, 3 ] ], type= object
6   --------------------essai n° 2
7   erreur=[ { nom: 'hercule', pays: 'grèce antique' } ], type= object
8   --------------------essai n° 3
9   erreur=[ Error: erreur de connexion au réseau
10  at Object.<anonymous> (c:\Data\st-2019\dev\es6\javascript\exceptions\excep-02.js:35:9)
11  at Object.<anonymous> (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:251206)
12  at c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:245054
13  at Generator.next (<anonymous>)
14  at bl (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:245412)
15  at kl (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:247659)
16  at Object.u (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:287740)
17  at Object.o (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:287137)
18  at Object.<anonymous> (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:284879)
19  at Object.apply (c:\Temp\19-09-01\javascript\node_modules\esm\esm.js:1:199341) ], type= object
20  --------------------essai n° 4
21  erreur.message=[ erreur de connexion au réseau ], type(error)= object
```

2.9.3 script [excep-03]

Ce script montre qu'on peut, dans un **[catch]**, différencier le type d'instance **[Error]** interceptée par le **[catch]** :

```
1.  'use strict';
2.
3.  // package moment
4.  import moment from 'moment';
5.
6.  // différencier l'instance d'Error reçue dans un [catch]
7.  for (let i = 0; i < 10; i++) {
8.    // date - heure du moment courant
9.    const now = Date.now();
10.   // formatage heure pour avoir les millisecondes
11.   const time = moment(now).format("HH:mm:ss:SSS");
12.   // les millisecondes
13.   const milli = Number(time.substr(time.length - 3));
14.   console.log("--------------------itération n° ", i);
15.   try {
16.     // nbre aléatoire
17.     const nbre = milli % 3;
18.     switch (nbre) {
19.       case 0:
20.         throw new ReferenceError("erreur 1");
21.       case 1:
22.         throw new RangeError("erreur 2");
23.       default:
24.         throw new EvalError("erreur 3");
25.     }
26.   } catch (error) {
27.     // il y a eu erreur
28.     if (error instanceof ReferenceError) {
29.       console.log("ReferenceError :", error.message);
30.     } else {
31.       if (error instanceof RangeError) {
32.         console.log("RangeError :", error.message);
33.       }
34.       else {
35.         if (error instanceof EvalError) {
36.           console.log("EvalError :", error.message);
37.         }
38.       }
39.     }
40.   }
41. }
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\exceptions\excep-03.js"
2   --------------------itération n° 0
3   ReferenceError : erreur 1
4   --------------------itération n° 1
5   RangeError : erreur 2
6   --------------------itération n° 2
7   RangeError : erreur 2
8   --------------------itération n° 3
9   RangeError : erreur 2
10  --------------------itération n° 4
11  RangeError : erreur 2
12  --------------------itération n° 5
13  RangeError : erreur 2
14  --------------------itération n° 6
15  EvalError : erreur 3
16  --------------------itération n° 7
17  EvalError : erreur 3
18  --------------------itération n° 8
19  EvalError : erreur 3
20  --------------------itération n° 9
21  EvalError : erreur 3
```

2.10 Les modules ES6

Les modules ES6 permettent de construire des applications Javascript structurées en modules indépendants et réutilisables.

2.10.1 scripts `[import-01, export-01]`

Le script **`[import-01]`** va utiliser le module **`[export-01]`**. Celui-ci est défini de la façon suivante :

```
1.  // export par défaut d'un objet non nommé
2.  export default {
3.    data: 2,
4.    do() {
5.      console.log(this.data);
6.    }
7.  };
```

Commentaires

- les lignes **`[2-7]`** définissent un objet non nommé avec les propriétés **`[data, do]`** ;
- ligne 2 : l'instruction **`[export default]`** exporte cet objet. Celui-ci pourra donc être importé ;

Le module **`[export-01]`** est utilisé par le script **`[import-01]`** de la façon suivante :

```
1.  'use strict';
2.  // import d'un objet xporté par défaut
3.  import export01 from './export-01';
4.  // utilisation de cet objet
5.  export01.do();
6.  // on peut importer un export par défaut sous n'importe quel nom
7.  import data from './export-01';
8.  console.log(data.data);
```

Commentaires

- les lignes 3 et 7 importent l'objet exporté par défaut du module **`[export-01]`** sous deux noms différents ;
- une fois un objet importé, on peut l'utiliser comme s'il avait été défini localement dans le script ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\modules\import-01.js"
2   2
3   2
```

2.10.2 scripts `[import-02, export-02]`

Ces scripts montrent un export d'un objet nommé.

Le script **`[export-02]`** est le suivant :

```
1.  // export par défaut d'un objet nommé
2.  const data = {
3.    data: 2,
4.    do() {
5.      console.log(this.data);
6.    }
7.  };
```

```
8.  // export
9.  export default data;
```

- ligne 9 : on exporte l'objet **[data]** ;

Que l'objet exporté soit nommé ou non ne change rien à l'opération d'import. Le script **[import-02]** est le suivant :

```
1.  'use strict';
2.  // import d'un objet exporté par défaut
3.  import module1 from './export-02';
4.  // utilisation de cet objet
5.  module1.do();
6.  // on peut importer un export par défaut sous n'importe quel nom
7.  import module2 from './export-02';
8.  console.log(module2.data);
```

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\modules\import-02.js"
2   2
3   2
```

2.10.3 scripts [import-03, export-03]

Un module peut exporter plusieurs éléments.

Le script **[export-03]** est le suivant :

```
1.  // multi-exports
2.  // export objet
3.  const data = {
4.    data: 2,
5.    do() {
6.      console.log(this.data);
7.    }
8.  };
9.  // export fonction
10. export { data };
11. function doSomething() {
12.   console.log("doSomething");
13. }
14. export { doSomething };
```

Commentaires

- lignes 10 et 14 : export de deux éléments. L'export d'un élément se fait avec la syntaxe **[export {élément}]** ;

Le script **[import-03]** utilise le module **[export-03]** de la façon suivante :

```
1.
2.  'use strict';
3.  // import d'un module [export03]
4.  import {data, doSomething} from './export-03';
5.  // utilisation des imports
6.  data.do();
7.  doSomething();
8.  // autre écriture
9.  import * as module from './export-03';
10. // utilisation de l'import
11. console.log(module.data);
12. module.doSomething();
```

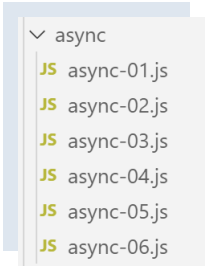
- ligne 3 : les **[imports]** se font avec les noms exacts des objets exportés ;
- ligne 8 : on peut importer tous les objets exportés dans un objet nommé (ici **[module]**) ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\modules\import-03.js"
2   2
```

```
3   doSomething
4   { data: 2, do: [Function: do] }
5   doSomething
```

2.11 Programmation événementielle et fonctions asynchrones



Une fonction asynchrone est une fonction dont l'exécution est lancée mais dont on n'attend pas le résultat. Lorsque l'exécution est terminée, la fonction asynchrone émet un événement et transmet son résultat via celui-ci.

Ce mode de fonctionnement est bien adapté à l'exécution au sein d'un navigateur web. En effet, une application exécutée au sein d'un navigateur est une application à événements : l'application réagit à des événements, principalement provoqués par l'utilisateur (clics, déplacements de souris, frappe de texte, …). Les applications Javascript exécutées au sein d'un navigateur sont amenées à dialoguer avec des services externes via le protocole HTTP. Les fonctions natives HTTP de Javascript sont asynchrones : elles sont lancées et l'obtention de la réponse du service externe sollicité est signalé par un événement qui s'ajoute à l'ensemble des événements gérés par l'application.

Les scripts suivants vont être exécutés par **[node.js]** et non par un navigateur. **[node.js]** a lui également un mode d'exécution par événement :

- **[node.js]**, comme un navigateur, utilise une boucle d'événements pour exécuter un script ;
- l'exécution du code principal du script est le 1er événement exécuté ;
- si ce code principal a lancé des tâches asynchrones alors l'exécution se poursuit tant que ces tâches asynchrones ne sont pas terminées. Celles-ci émettent un événement lorsqu'elles sont terminées. Ces événements sont mis en file d'attente dans la boucle d'événements ;
- le script principal doit s'abonner à ces événements s'il veut récupérer les résultats des actions asynchrones ;
- le script n'est terminé que lorsque tous les événements émis par celui-ci ont été traités ;

2.11.1 script [async-01]

Le script suivant montre le comportement d'un script comportant une action asynchrone.

```
1.  'use strict';
2.
3.  // imports
4.  import moment from 'moment';
5.  import { sprintf } from 'sprintf-js';
6.
7.  // début
8.  const débutScript = moment(Date.now());
9.  console.log("[début du script],", heure());
10.
11. // setTimeout arme un timer de 1000 ms (2ième paramètre) et retourne immédiatement le n° de ce timer
12. // lorsque le timer a épuisé les 1000 ms il émet un événement qui est mis en file d'attente du runtime
13. // lorsque l'événement est traité par le runtime, la fonction (1er paramètre) est exécutée
14. setTimeout(function () {
15.   // ce code sera exécuté lorsque le timer aura atteint la valeur 0
16.   console.log("[fin de l'action asynchrone setTimeout],", heure(débutScript));
17. }, 1000)
18.
19. // s'affichera avant le msg de la fonction interne au timer
20. console.log("[fin du code principal du script],", heure(débutScript));
21.
22. // utilitaire d'affichage heure et durée
23. function heure(début) {
24.   // heure du moment courant
25.   const now = moment(Date.now());
26.   // formatage heure
27.   let result = "heure=" + now.format("HH:mm:ss:SSS");
28.   // faut-il calculer une durée ?
```

```
29.   if (début) {
30.     const durée = now - début;
31.     const milliseconds = durée % 1000;
32.     const seconds = Math.floor(durée / 1000);
33.     // formatage heure + durée
34.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds, milliseconds);
35.   }
36.   // résultat
37.   return result;
38. }
```

- ligne 4 : on importe la bibliothèque **[moment]** pour pouvoir formater les dates (ligne 27) ;
- ligne 5 : on importe la bibliothèque **[sprintf-js]** pour pouvoir formater les durées (ligne 34) ;
- ligne 8 : on note l'heure de début du script ;
- ligne 9 : on l'affiche à l'aide de la méthode **[heure]** des lignes 20-34 ;
- lignes 14-17 : la fonction **[setTimeout]** a deux paramètres (f, durée) : **f** est une fonction qui est exécutée lorsque **[durée]** millisecondes se sont écoulées ;
- ligne 14 : à l'exécution du script, la fonction **[setTimeout]** est exécutée :
  - ligne 17 : un minuteur de 1000 ms est armé et le décompte commence jusqu'à atteindre ultérieurement 0. La fonction **[setTimeout]** est terminée dès que le minuteur est initialisé et le décompte commencé. Elle **n'attend pas** la fin de ce décompte. Elle rend un n° identifiant le minuteur utilisé et l'exécution passe à l'instruction suivante, ligne 20. Ici, le résultat de **[setTimeout]** n'est pas utilisé ;
- ligne 16 : ce message va s'afficher à la fin du délai de 1000 ms de la fonction **[setTimeout]** ;
- lignes 15-16 : la fonction f, 1er paramètre de la fonction **[setTimeout]**, va être exécutée à la fin du délai des 1000 ms. Le message de la ligne 16 va alors s'afficher ;
- ligne 20 : ce message va s'afficher avant celui de la ligne 16 ;

Fonction **heure** :

- ligne 23 : la fonction admet un paramètre facultatif **[heure]** qui est l'heure de début d'une opération dont elle doit afficher la durée ;
- lignes 25-27 : on calcule et formate l'heure du moment ;
- ligne 29 : si le paramètre **[début]** est présent alors il faut calculer une durée ;
- ligne 30 : la durée de l'opération. On obtient un nombre de millisecondes ;
- lignes 31-32 : ce nombre de millisecondes est décomposé en secondes et millisecondes ;
- ligne 34 : la durée est ajoutée à l'heure ;

Exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\async\async-01.js"
2   [début du script], heure=09:26:40:238
3   [fin du code principal du script], heure=09:26:40:246, durée= 0 seconde(s) et 11 millisecondes
4   [fin de l'action asynchrone setTimeout], heure=09:26:41:249, durée= 1 seconde(s) et 14 millisecondes
5
6   [Done] exited with code=0 in 1.672 seconds
```

- ligne 4, on voit que l'action asynchrone **[setTimeOut]** s'est terminée 1s environ après la fin du code principal du script ;
- ligne 6 : l'heure affichée ligne 3 est celle de la fin du code principal. Si celui-ci a lancé des tâches asynchrones, le script n'est terminé que lorsque toutes les tâches asynchrones ont été exécutées. La durée affichée ligne 6, est la durée totale d'exécution du script (code principal + tâches asynchrones) ;

La fonction **[setTimeout]** va nous permettre de simuler des tâches asynchrones dans un environnement **[node.js]**. En effet, la fonction **[setTimeout]** se comporte comme une tâche asynchrone :

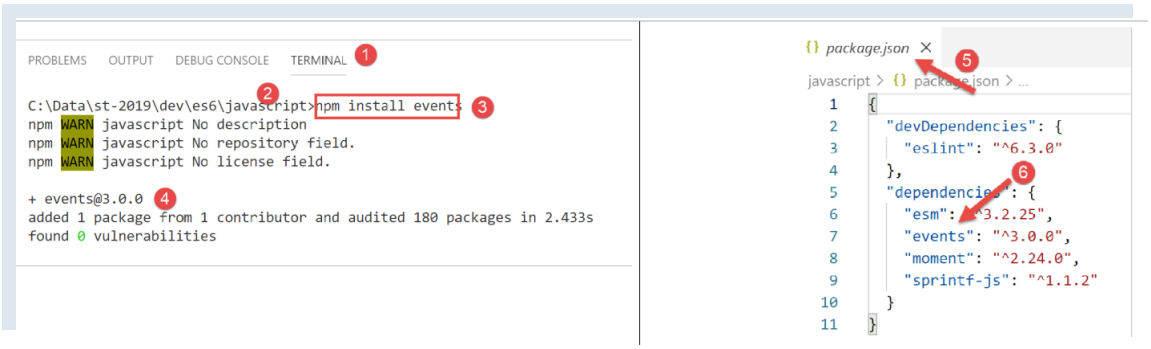
- elle rend un résultat immédiatement, ici un n° de timer, par le mécanisme usuel des fonctions (return) ;
- elle peut rendre **ultérieurement** (ce n'est pas encore le cas ci-dessus) d'autres résultats **via des événements** qui sont alors traités par la boucle d'événements de **[node.js]** ;
- dans la plupart des cas qui vont suivre, ces événements seront au nombre de deux :
  - un événement qu'on pourrait appeler **[success]** qui sera émis par la tâche asynchrone qui a réussi ce qu'elle devait faire. Une donnée, le résultat de la tâche, est associée à l'événement émis ;
  - un événement qu'on pourrait appeler **[failure]** qui sera émis par la tâche asynchrone qui a échoué à faire ce qu'elle devait faire. Une donnée, un objet décrivant l'erreur généralement, est associé à l'événement émis. Des erreurs possibles par exemple avec une tâche internet asynchrone seraient 'réseau indisponible', 'machine serveur inexistante', 'timeout dépassé', ...

- le code principal qui a lancé une tâche asynchrone peut s'abonner aux événements que cette tâche est susceptibe d'émettre. Lorsqu'un de ceux-ci est émis, le code principal en est averti et peut déclencher l'exécution d'une fonction particulière destinée à traiter l'événement. Cette fonction reçoit en paramètre, la donnée que la tâche asynchrone a associée à l'événement émis ;

2.11.2 script [async-02]

Dans ce script, la fonction asynchrone **[setTimeout]** va émettre des événements pour communiquer des données aux codes qui se seront abonnés à ceux-ci.

L'accès aux événements de **[node.js]** nécessite des bibliohèques supplémentaires. Nous choisissons la bibliothèque **[events]** que nous installons avec **[npm]** :



Le script **[async-02]** est le suivant :

```
1.  'use strict';
2.
3.  // les fonctions asynchrones peuvent rendre un résultat en émettant un événement
4.  // le code principal peut récupérer ces résultats en s'abonnant aux événements émis
5.
6.  // imports
7.  import moment from 'moment';
8.  import { sprintf } from 'sprintf-js';
9.  import EventEmitter from 'events';
10.
11. // début
12. const débutScript = moment(Date.now());
13. console.log("[début du script],", heure());
14. // un émetteur d'événements
15. const eventEmitter = new EventEmitter();
16.
17. // setTimeout arme un timer de 1000 ms (2ième paramètre) et retourne immédiatement le n° de ce timer
18. // lorsque le timer a épuisé les 1000 ms il émet un événement qui est mis en file d'attente du runtime
19. // lorsque l'événement est traité par le runtime, la fonction (1er paramètre) est exécutée
20. setTimeout(function () {
21.   // ce code sera exécuté lorsque le timer aura atteint la valeur 0
22.   console.log("[setTimeout, fin du timer d'1 s],", heure(débutScript));
23.   // on émet un événement pour dire qu'un résultat est disponible
24.   eventEmitter.emit("timer1Success", { success: 4 });
25.   // on émet un autre événement pour dire qu'un autre résultat est disponible
26.   eventEmitter.emit("timer1Failure", { failure: 6 });
27. }, 1000)
28.
29. // on s'abonne à l'évt [timer1Success]
30. eventEmitter.on('timer1Success', (result) => {
31.   console.log(sprintf("la fonction asynchrone du timer a rendu le résultat [%j], %s, via l'événement [timer
    1Success]", result, heure(débutScript)));
32. });
33.
34. // on s'abonne à l'évt [timer1Failure]
35. eventEmitter.on('timer1Failure', (result) => {
36.   console.log(sprintf("la fonction asynchrone du timer a rendu le résultat [%j], %s, via l'événement [timer
    1Failure]", result, heure(débutScript)));
37. });
38.
39. // s'affichera avant les msg des evts émis par la fonction associée à [timer1]
40. console.log("[fin du code principal du script],", heure(débutScript));
41.
42. // utilitaire d'affichage heure et durée
43. function heure(début) {
```

```
44.   // heure du moment courant
45.   const now = moment(Date.now());
46.   // formatage heure
47.   let result = "heure=" + now.format("HH:mm:ss:SSS");
48.   // faut-il calculer une durée ?
49.   if (début) {
50.     const durée = now - début;
51.     const milliseconds = durée % 1000;
52.     const seconds = Math.floor(durée / 1000);
53.     // formatage heure + durée
54.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds, milliseconds);
55.   }
56.   // résultat
57.   return result;
58. }
```

Commentaires

- ligne 9, on importe la classe **[EventEmitter]** de la bibliothèque **[events]**. C'est une nouveauté : nous n'avions jusqu'à maintenant importé que des objets littéraux et des fonctions ;
- ligne 15 : on crée un émetteur d'événements **[node.js]** en instanciant la classe **[EventEmitter]** avec le mot clé **[new]** ;
- lignes 20-27 : la fonction asynchrone **[setTimeout]**. Elle va émettre deux événements lors de son exécution :
  - ligne 24, l'événement **[timer1Success]** avec comme valeur associée l'objet {success : 4} ;
  - ligne 26, l'événement **[timer1Failure]** avec comme valeur associée l'objet {failure : 6} ;
  - une fonction asynchrone peut émettre autant d'événements qu'elle veut. On a dit précédemment que le plus souvent elle émettait l'un des deux événements **[success, failure]**, pas les deux comme on le fait ici ;
- ligne 20 : l'exécution de **[setTimeout]** est instantanée : un timer est armé et le n° de celui-ci rendu au code appelant. L'émission des événements se fera plus tard, ici 1 seconde plus tard ;
- l'émission d'événements est inutile s'il n'y a aucun code pour les exploiter lorsqu'ils surviendront. C'est pourquoi le code principal doit s'abonner aux deux événements **[timer1Success, timer1Failure]** s'il veut les gérer, notamment récupérer les données associées à ces événements ;
- lignes 30-32 : le code principal s'abonne à l'événement **[timer1Success]**. Lorsque la bouche d'événements de **[node.js]** traitera cet événement, il appellera la fonction qui est le second paramètre de la méthode **[eventEmitter.on]** en lui passant la donnée (ici appelée **[result]**) associée à l'événement **[timer1Success]** ;
- ligne 31 : la fonction de traitement de l'événement affichera le jSON de la donnée associée à l'événement ainsi que l'heure du moment ;
- lignes 35-37 : avec un code analogue, le code principal s'abonne à l'événement **[timer1Failure]** ;
- l'abonnement à un événement (1er paramètre) n'exécute pas immédiatement le code de la fonction **[callback]** (2ième paramètre). Celle-ci ne sera exécutée qu'après que l'événement ait eu lieu ;
- ligne 40 : le code principal du script est terminé mais pas le script lui-même puisque le code principal a lancé une tâche asynchrone. Le script global ne sera terminé qu'après la fin de cette tâche asynchrone ;

C'est ce que montrent les résultats obtenus :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\async\async-02.js"
2   [début du script], heure=09:34:58:909
3   [fin du code principal du script], heure=09:34:58:916, durée= 0 seconde(s) et 10 millisecondes
4   [setTimeout, fin du timer d'1 s], heure=09:34:59:929, durée= 1 seconde(s) et 23 millisecondes
5   la fonction asynchrone du timer a rendu le résultat [{"success":4}], heure=09:34:59:931, durée= 1 seconde(s
    ) et 25 millisecondes, via l'événement [timer1Success]
6   la fonction asynchrone du timer a rendu le résultat [{"failure":6}], heure=09:34:59:932, durée= 1 seconde(s
    ) et 26 millisecondes, via l'événement [timer1Failure]
7
8   [Done] exited with code=0 in 1.627 seconds
9
```

- ligne 3 : fin du code principal 10 ms après le début du script ;
- ligne 4 : début de la fonction encapsulée dans le timer de 1000 ms, 1 seconde environ après le début du script ;
- ligne 5 : traitement de l'événement **['timer1Success']**, 2 ms plus tard ;
- ligne 6 : traitement de l'événement **['timer1Failure']**, 1 ms plus tard que l'événement **['timer1Success']** ;
- ligne 8 : fin du script global avec une durée totale de 1,627 seconde ;

2.11.3 script [async-03]

Le script suivant montre un autre aspect de la boucle événementielle de **[node.js]** :

- la boucle exécute les événements les uns après les autres, généralement dans leur ordre d'arrivée. Certains OS accordent des priorités aux événements qui sont alors traités par ordre de priorité et non par ordre d'arrivée ;

- la boucle **n'exécute qu'un événement à la fois**. Le suivant n'est traité que lorsque le traitement du précédent est terminé. Dans un système événementiel, il faut donc éviter d'écrire du code qui monopolise longtemps le processeur car alors les événements ne sont pas traités lorsqu'ils se produisent mais plus tard lorsque la boucle événementielle arrive à eux. On a alors une application peu « réactive » ;

Le script **[async-03]** montre un exemple de ce phénomène :

```
1.  'use strict';
2.
3.  // les fonctions asynchrones peuvent rendre un résultat en émettant un événement
4.  // le code principal peut récupérer ces résultats en s'abonnant aux événements émis
5.
6.  // imports
7.  import moment from 'moment';
8.  import { sprintf } from 'sprintf-js';
9.  import EventEmitter from 'events';
10.
11. // début
12. const débutScript = moment(Date.now());
13. console.log("[début du script],", heure());
14. // un émetteur d'événements
15. const eventEmitter = new EventEmitter();
16.
17. // setTimeout arme un timer de 1000 ms (2ième paramètre) et retourne immédiatement le n° de ce timer
18. // lorsque le timer a épuisé les 1000 ms il émet un événement qui est mis en file d'attente du runtime
19. // lorsque l'événement est traité par le runtime, la fonction (1er paramètre) est exécutée
20. setTimeout(function () {
21.   // ce code sera exécuté lorsque le timer aura atteint la valeur 0
22.   console.log("[setTimeout, fin du timer d'1 s],", heure(débutScript));
23.   // on émet un événement pour dire qu'un résultat est disponible
24.   eventEmitter.emit("timer1Success", { success: 4 });
25.   // on émet un autre événement pour dire qu'un autre résultat est disponible
26.   eventEmitter.emit("timer1Failure", { failure: 6 });
27. }, 1000)
28.
29. // on s'abonne à l'évt [timer1Success]
30. eventEmitter.on('timer1Success', (result) => {
31.   console.log(sprintf("la fonction asynchrone du timer a rendu le résultat [%j], %s, via l'événement [timer
    1Success]", result, heure(débutScript)));
32. });
33.
34. // on s'abonne à l'évt [timer1Failure]
35. eventEmitter.on('timer1Failure', (result) => {
36.   console.log(sprintf("la fonction asynchrone du timer a rendu le résultat [%j], %s, via l'événement [timer
    1Failure]", result, heure(débutScript)));
37. });
38.
39. // un code synchrone un peu intensif qui a empêcher le code principal de s'achever avant la fin de [timer1]
40. for (let i = 0; i < 1000000; i++) {
41.   for (let j = 0; j < 10000; j++) {
42.     i + i ^ 2 + i ^ 3;
43.   }
44. }
45.
46. // s'affichera avant les msg des evts émis par la fonction associée à [timer1]
47. console.log("[fin du code principal du script],", heure(débutScript));
48.
49. // utilitaire d'affichage heure et durée
50. function heure(début) {
51.   …
52. }
```

Commentaires

- ce code est celui de l'exemple précédent **[async-02]** auquel on a ajouté les lignes 39-44 ;
- lignes 20-27 : la fonction **[setTimeout]** a été programmée pour exécuter une fonction asynchrone interne au bout d'un délai d'une seconde. Au bout de cette seconde, l'exécution de la fonction asynchrone du timer n'a pas lieu immédiatement : un événement est placé dans la boucle d'exécution pour demander celle-ci. Si la boucle d'exécution est occupée à traiter un autre événement, l'exécution de la fonction asynchrone du timer devra attendre ;
- lignes 20-27 : dès que la la fonction **[setTimeout]** a armé son timer d'un délai d'une seconde, elle lâche le processeur et rend la main au code appelant. Celui-ci continue avec les 30-37 qui sont des abonnements à des événements et qui ont un temps d'exécution négligeable ;

- le code principal continue avec les lignes 40-44 qui forment une boucle de 10^{10} itérations. Ce code sera en cours d'exécution lorsque le timer va émettre son événement de « fin du délai d'1 seconde ». Cet événement est alors mis dans la boucle événementielle mais devra attendre la fin d'exécution du code principal du script pour avoir une chance d'être traité ;
- ligne 47 : fin du code principal du script. C'est après ce dernier affichage que l'événement de fin de timer va pouvoir être traité et la fonction asynchrone interne à **[setTimeout]** va pouvoir être exécutée ;

Le script donne les résultats suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\async\async-03.js"
2   [début du script], heure=08:55:02:665
3   [fin du code principal du script], heure=08:55:11:789, durée= 9 seconde(s) et 131 millisecondes
4   [setTimeout, fin du timer d'1 s], heure=08:55:11:794, durée= 9 seconde(s) et 136 millisecondes
5   la fonction asynchrone du timer a rendu le résultat [{"success":4}], heure=08:55:11:794, durée= 9 seconde(s
    ) et 136 millisecondes, via l'événement [timer1Success]
6   la fonction asynchrone du timer a rendu le résultat [{"failure":6}], heure=08:55:11:794, durée= 9 seconde(s
    ) et 136 millisecondes, via l'événement [timer1Failure]
7
8   [Done] exited with code=0 in 9.796 seconds
```

Commentaires

- ligne 3 : on voit que le code principal du script a mis 9 secondes à s'exécuter. Les événements qui ont pu se produire pendant ce temps ont été mis en attente dans la boucle événementielle ;
- ligne 4 : on voit que l'événement **[fin du timer]** a été traité 5 ms après la fin du code principal. Il a été émis environ 1 s après le début du script mais a du attendre 8s supplémentaires pour être finalement traité ;

On retiendra de cet exemple que dans un système événementiel, un code ne doit jamais occuper le processeur très longtemps. Si on a un code synchrone long à exécuter, on doit se « débrouiller » pour le décomposer en tâches asynchrones plus courtes qui signaleront leur fin avec un événement.

2.11.4 script [async-04]

Le script **[async-04]** montre un autre mécanisme, appelé **[Promise]**, une promesse de résultat. Ce mécanisme évite de gérer explicitement des événements **[node.js]**. C'est fait implicitement et le développeur peut alors ignorer l'existence de ces événements. Les comprendre lui permettra cependant de mieux appréhender le fonctionnement des **[Promise]** qui est de prime abord complexe.

Le type **[Promise]** est une classe Javascript. Son constructeur admet comme paramètre une fonction asynchrone à qui elle passe deux paramètres appelés traditionnellement **[resolve]** et **[reject]**. Ils pourraient porter un autre nom ;

```
1.  const promise=new Promise(function(resolve, reject){
2.          // une tâche asynchrone est lancée
3.          …
4.          // si réussite : appeler resolve(result) où [result] est le résultat de la tâche asynchrone ;
5.          // si échec : appeler reject(error) où [error] est un objet encapsulant l'erreur rencontrée ;
6.  }
7.  // s'abonner aux événements émis par la tâche asynchrone de la [Promise]
```

- le constructeur de **[Promise]** fait deux choses :
  - il crée un événement pour lancer l'exécution de la fonction **[function(resolve, reject)]** qu'on lui a passée en paramètre mais n'attend pas son résultat et rend immédiatement un objet **[Promise]** au code appelant. Celui-ci peut avoir quatre états :
    - **[pending]** : l'action asynchrone qui a rendu la **[Promise]** n'est pas encore terminée ;
    - **[fulfilled]** : l'action asynchrone qui a rendu la **[Promise]** s'est terminée avec succès ;
    - **[rejected]** : l'action asynchrone qui a rendu la **[Promise]** s'est terminée sur un échec ;
    - **[settled]** : l'action asynchrone qui a rendu la **[Promise]** est terminée ;
      - Lorsque le constructeur rend son résultat, l'objet **[Promise]** créé est dans l'état **[pending]**, en attente des résultats de la fonction asynchrone ;
  - la tâche asynchrone des lignes 2-5 est **lancée immédiatement**. Les tâches asynchrones sont le plus souvent des tâches asynchrones d'entrée / sortie qui se décomposent de la façon suivante :
    - exécution d'un code synchrone pour lancer l'opération d'E/S avec un autre organe, par exemple un serveur distant ;
    - attente de la réponse de cet organe ;
    - traitement de cette réponse ;

    C'est la phase 2 d'attente de l'organe extérieur au processeur qui est le plus coûteux. Plutôt que d'attendre :

- la réception de la donnée demandée à l'organe extérieur va être signalée par un événement ;

- dans le code synchrone qui va suivre la phase 1 (ligne 7 du code exemple), on va s'abonner à cet événement puis à un moment retourner dans la boucle d'événements de **[node.js]**. L'événement suivant dans la liste des événements en attente va alors être traité ;
- pendant la phase 2, il y a parallélisme d'exécution mais sur des périphériques différents :
  - le processeur pour la boucle d'événements ;
  - un organe extérieur (disque, base de données, serveur distant) pour la recherche de la donnée demandée ;
- à la fin de la phase 2, lorsque l'opération d'E/S a obtenu la donnée qu'elle demandait, un événement va être émis pour indiquer que le résultat de l'E/S est disponible. Cet événement va alors rejoindre les autres dans la liste d'attente des événements ;
- lorsque son tour viendra, il sera traité. La fonction associée à cet événement (ligne 7 du code exemple) va alors être exécutée ;

    Ce mode de fonctionnement permet d'éviter les temps morts : celui où le processeur attend la réponse d'un périphérique plus lent que lui ;

  - lorsque la tâche asynchrone des lignes 2 et 5 a été lancée et a terminé son travail, elle a la possibilité de rendre un résultat au code appelant, grâce aux deux fonctions **[resolve, reject]** que le constructeur **[Promise]** lui a passé en paramètres. La convention est la suivante :

    - la tâche asynchrone signale un succès par **[resolve(result)]**. Cela revient à mettre dans la boucle d'événements de **[node.js]**, un événement qu'on pourrait appeler **[resolved]** avec **[result]** comme donnée associée ;
    - la tâche asynchrone signale un échec par **[reject(error)]**. Cela revient à mettre dans la boucle d'événements de **[node.js]**, un événement qu'on pourrait appeler **[rejected]** avec **[error]** comme donnée associée, en général un objet détaillant l'erreur qui s'est produite ;
    - il faut donc que le code appelant s'abonne à ces deux événements pour être prévenu de la disponibilité du résultat de la fonction asynchrone ;

Après l'exécution terminée de la tâche asynchrone encapsulée dans la **[Promise]**, l'état de l'objet **[promise]** rendu par le constructeur **[Promise(…)]** change :

- l'événement **[resolved]** le fait passer de l'état **[pending]** à **[resolved]** ;
- l'événement **[rejected]** le fait passer de l'état **[pending]** à **[rejected]** ;

L'abonnement aux événements **[resolved]** et **[rejected]** de la tâche asynchrone se fait avec des méthodes de la classe **[Promise]** avec la syntaxe suivante :

```
promise.then(f1).catch(f2).finally(f3) ;
```

où :

- **f1** est une fonction exécutée lorsque l'état de **[promise]** passe de **[pending]** à **[resolved]**, donc lorsque la tâche asynchrone a réussi son travail. Elle reçoit pour paramètre la valeur **[result]**, transmise par l'instruction **[resolve(result)]** de la tâche asynchrone ;
- **f2** est une fonction exécutée lorsque l'état de **[promise]** passe de **[pending]** à **[rejected]**, donc lorsque la tâche asynchrone a échoué à faire son travail. Elle reçoit pour paramètre la valeur **[error]**, transmise par l'instruction **[reject(errror)]** de la tâche asynchrone ;
- **f3** est une fonction exécutée après exécution des méthodes **[then]** ou **[catch]**, donc tout le temps exécutée. Elle ne reçoit aucun paramètre ;

Cette syntaxe cache complètement les événements auxquels on s'abonne. C'est pourtant un abonnement et comme celui de l'exemple précédent, il n'exécute pas immédiatement les fonctions **[f1, f2, f3]**. Celles-ci seront exécutées ou pas lorsque l'un des événements **[resolved, rejected]** auxquels on s'abonne va se produire.

Le script **[async-04]** montre cette mécanique :

```
1.  'use strict';
2.
3.  // il est possible d'obtenir les résultats (succes, failure) d'une fonction asynchrone
4.  // sans utiliser explicitement des événements grâce à la classe [Promise]
5.  // cette classe utilise implicitement des événements mais ceux-ci ne se voient pas dans le code
6.
7.  // imports
8.  import moment from 'moment';
9.  import { sprintf } from 'sprintf-js';
10.
11. // début
12. const débutScript = moment(Date.now());
13. console.log("[début du script],", heure(débutScript));
14.
15. // définition d'une tâche asynchrone à l'aide d'une promesse [Promise]
```

```
16. // la tâche asynchrone est le paramètre du constructeur [Promise]
17. const débutPromise1 = moment(Date.now());
18. const promise1 = new Promise(function (resolve) {
19.   // log
20.   console.log("[début fonction asynchrone de promise1],", heure(débutPromise1));
21.   // code asynchrone
22.   setTimeout(function () {
23.     console.log("[fin fonction asynchrone de promise1],", heure(débutPromise1));
24.     // la tâche asynchrone rend un résultat avec la fonction [resolve]
25.     // la promesse est alors réussie
26.     resolve('[réussite]');
27.   }, 1000)
28. });
29.
30. // on peut connaître le résultat de la promesse [promise1]
31. // lorsque celle-ci a été résolue (resolve) ou rejetée (reject)
32. // l'instruction qui suit est un abonnement à l'évt [resolved] via la méthode [then]
33. // et à l'évt [rejected] via la méthode [catch]
34. // la méthode [finally] est exécutée que ce soit après un then ou un catch
35. promise1.then(result => {
36.   // cas de réussite de la promesse  [evt resolved]
37.   console.log(sprintf("[promise1.then], %s, result=%s", heure(débutPromise1), result));
38. }).catch(result => {
39.   // cas d'erreur  [evt rejected]
40.   console.log(sprintf("[promise1.catch], %s, result=%s", heure(débutPromise1), result));
41. }).finally(() => {
42.   // exécuté dans tous les cas
43.   console.log("[promise1.finally]", heure(débutPromise1));
44. });
45.
46. // définition d'une tâche asynchrone à l'aide d'une promesse [Promise]
47. const débutPromise2 = moment(Date.now());
48. const promise2 = new Promise(function (resolve, reject) {
49.   // log
50.   console.log("[début fonction asynchrone de promise2],", heure(débutPromise2));
51.   // tâche asynchrone
52.   setTimeout(function () {
53.     console.log("[fin fonction asynchrone de promise2],", heure(débutPromise2));
54.     // la tâche asynchrone rend un résultat avec la fonction [reject]
55.     // la promesse est alors ratée
56.     reject('[échec]');
57.   }, 2000)
58. });
59.
60. // on peut connaître le résultat de la promesse [promise2]
61. // lorsque celle-ci a été résolue (resolve) ou rejetée (reject)
62. promise2.then(result => {
63.   // cas de réussite de la promesse [evt resolved]
64.   console.log(sprintf("[promise2.then], %s, result=%s", heure(débutPromise2), result));
65. }).catch(result => {
66.   // cas d'erreur [evt rejected]
67.   console.log(sprintf("[promise2.catch], %s, result=%s", heure(débutPromise2), result));
68. }).finally(() => {
69.   // exécuté dans tous les cas
70.   console.log(sprintf("[promise2.finally], %s", heure(débutPromise2)));
71. });
72.
73. // s'affichera avant les msg des fonctions asynchrones et ceux des évts associés
74. console.log("[fin du code principal du script],", heure(débutScript));
75.
76. // utilitaire
77. function heure(début) {
78.   // heure du moment courant
79.   const now = moment(Date.now());
80.   // formatage heure
81.   let result = "heure=" + now.format("HH:mm:ss:SSS");
82.   if (début) {
83.     const durée = now - début;
84.     const milliseconds = durée % 1000;
85.     const seconds = Math.floor(durée / 1000);
86.     // formatage durée
87.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds, milliseconds);
88.   }
89.   // résultat
90.   return result;
91. }
```

Commentaires

- lignes 18-28 : création d'une **`[Promise promise1]`**. Sa fonction asynchrone rend son résultat via un événement au bout d'une seconde. Une fois cet opération asynchrone lancée (armement d'un timer), on n'attend pas que celle-ci rend son résultat et on passe tout de suite au code de la ligne 35 ;

- lignes 35-44 : on s'abonne aux deux événements **[resolved, rejected]** que la fonction asynchrone interne à **[promise1]** peut émettre ;
- lignes 46-71 : on répète la même séquence de code que précédemment pour une seconde promesse **[promise2]** ;
- ligne 74 : le code principal du script est terminé mais pas le script dans son ensemble car deux actions asynchrones ont été lancées. On retourne dans la boucle événementielle où à un moment l'un des événements **[resolved, rejected]** des promesses **[promise1, promise2]** va se produire. Il sera alors traité ;
- puis il y aura retour à la boucle événementielle. Et là le second événement **[resolved, rejected]** des promesses **[promise1, promise2]** sera traité lorsqu'il se produira ;

Exécution

```
1.  [Running] C:/MyPrograms/laragon/bin/nodejs/node-v12/node.exe -r esm "c:\Data\st-2020\dev\php7-
    ecmascript6-vuejs-nuxtjs\es6\async\async-04.js"
2.  [début du script], heure=15:15:16:884, durée= 0 seconde(s) et 9 millisecondes
3.  [début fonction asynchrone de promise1], heure=15:15:16:895, durée= 0 seconde(s) et 1 millisecondes
4.  [début fonction asynchrone de promise2], heure=15:15:16:897, durée= 0 seconde(s) et 1 millisecondes
5.  [fin du code principal du script], heure=15:15:16:897, durée= 0 seconde(s) et 22 millisecondes
6.  [fin fonction asynchrone de promise1], heure=15:15:17:900, durée= 1 seconde(s) et 6 millisecondes
7.  [promise1.then], heure=15:15:17:901, durée= 1 seconde(s) et 7 millisecondes, result=[réussite]
8.  [promise1.finally] heure=15:15:17:902, durée= 1 seconde(s) et 8 millisecondes
9.  [fin fonction asynchrone de promise2], heure=15:15:18:901, durée= 2 seconde(s) et 5 millisecondes
10. [promise2.catch], heure=15:15:18:902, durée= 2 seconde(s) et 6 millisecondes, result=[échec]
11. [promise2.finally], heure=15:15:18:902, durée= 2 seconde(s) et 6 millisecondes
12.
13. [Done] exited with code=0 in 2.532 seconds
```

Commentaires

- ligne 3 : la fonction asynchrone de **[promise1]** est lancée mais on n'attend pas sa fin qui sera signalée par un événement ;
- ligne 4 : la fonction asynchrone de **[promise2]** est lancée mais on n'attend pas sa fin qui sera signalée par un événement ;
- ligne 5 : fin du code principal et retour à la boucle événementielle ;
- ligne 6 : traitement de l'événement **[fin fonction asynchrone de promise1]**. L'état de **[promise1]** va passer à **[resolved]**. Un événement le signale ;
- ligne 7 : **[promise2]** n'ayant toujours pas fini son travail, l'événement **[promise1 resolved]** qui vient d'être mis dans la boucle va être traité par la méthode **[promise1.then]** puis par la méthode **[promise.finally]** (ligne 8) ;
- lignes 9-11 : le même mécanisme se déroule lorsque **[promise2]** passe de l'état **[pending]** à **[resolved]** ;

2.11.5 script [async-05]

Revenons au code du constructeur d'un objet **[Promise]** :

```
1.  const promise=new Promise(function(resolve, reject){
2.          // une tâche asynchrone est lancée
3.          …
4.          // si réussite : appeler resolve(result) où [result] est le résultat de la tâche asynchrone ;
5.          // si échec : appeler reject(error) où [error] est un objet encapsulant l'erreur rencontrée ;
6.  }
7.  // on s'abonne aux événements émis par la tâche asynchrone
```

Ligne 2, la tâche asynchrone de la **[Promise]** est lancée. Elle a souvent besoin de davantage de paramètres que les seuls paramètres **[resolve, reject]** que la fonction qui l'encapsule lui passe. Dans ce cas, on encapsule la création de la **[Promise]** dans une fonction qui va lui passer les paramètres dont sa fonction asynchrone a besoin :

```
1.  // définition de la fonction asynchrone
2.  function uneFonctionAsynchrone (p1, p2, …, pn){
3.   return new Promise(function(resolve, reject){
4.          // une tâche asynchrone est lancée avec les paramètres (P1, p2, …, pn)
5.          …
6.          // si réussite : appeler resolve(result) où [result] est le résultat de la tâche asynchrone ;
7.          // si échec : appeler reject(error) où [error] est un objet encapsulant l'erreur rencontrée ;
8.  }
9.  // on s'abonne aux évts [resolved, rejected] que va émettre la fonction asynchrone [uneFonctionAsynchrone]
10. …
11. // qq temps plus tard, la fonction asynchrone [uneFonctionAsynchrone] est appelée
12. uneFonctionAsynchrone(e1, e2, …, en) ;
```

Le script suivant :

- définit deux fonctions asynchrones rendant une **[Promise]** ;
- lance leur exécution en parallèle et attend que les deux soient terminées pour faire un certain travail ;

```
'use strict';

// on peut définir des fonctions asynchrones qui rendent un type [Promise]
// elles peuvent être alors taguées avec le mot clé [async]
// là encore
// imports
import moment from 'moment';
import { sprintf } from 'sprintf-js';

// début
const débutScript = moment(Date.now());
console.log("[début du script],", heure());

// une fonction asynchrone peut rendre une promesse [Promise]
// et avoir alors l'attribut [async]
async function async01(p1) {
  return new Promise((resolve) => {
    console.log("[début de la tâche asynchrone async01]");
    // la tâche asynchrone
    const débutAsync01 = moment(Date.now());
    setTimeout(function () {
      console.log("[fin de la tâche asynchrone async01],", heure(débutAsync01));
      // la tâche asynchrone peut rendre un résultat complexe
      resolve({
        prop1: [10, 20, 30],
        prop2: "abcd",
        prop3: p1,
      });
    }, 1000)
  });
}

// une fonction peut rendre une promesse [Promise]
// et peut alors avoir l'attribut [async]
async function async02(p1, p2) {
  return new Promise(resolve => {
    console.log("[début de la tâche asynchrone async02]");
    // tâche asynchrone
    const débutAsync02 = moment(Date.now());
    setTimeout(function () {
      console.log("[fin de la tâche asynchrone async02],", heure(débutAsync02));
      // la tâche asynchrone peut rendre un résultat complexe
      resolve({
        prop1: [11, 21, 31],
        prop2: "xyzt",
        prop3: p1 + p2
      });
    }, 2000)
  })
}

// on lance les deux fonctions asynchrones en parallèle
// et on attend qu'elles aient terminé toutes les deux
// le then ne s'exécutera que si les deux fonctions ont émis l'évt [resolved]
// le catch s'exécutera dès que l'une des deux fonctions émet l'évt [rejected]
Promise.all([async01(10), async02(10, 20)])
  // le résultat est un tableau [result1, result2] où [result1] est le résultat émis par un [resolve] de [async01]
  // et [result2] le résultat émis par un [resolve] de [async02]
  .then(result => {
    console.log(sprintf("[promise-all success], %s, result=%j", heure(débutScript), result));
  })
  // error est le résultat émis par le premier [reject] de l'une des deux fonctions asynchrones
  .catch(error => {
    console.log(sprintf("[promise-all error], %s, erreur=%j", heure(débutScript), error));
  })
  // finally est exécuté après le then ou le catch
  .finally(() => {
    console.log(sprintf("[promise-all finally], %s", heure(débutScript)));
  });

// s'affichera avant les msgs des fonctions asynchrones et des évts associés
console.log("[fin du code principal du script],", heure(débutScript));

// utilitaire
function heure(début) {
  // heure du moment courant
  const now = moment(Date.now());
  // formatage heure
  let result = "heure=" + now.format("HH:mm:ss:SSS");
  if (début) {
    const durée = now - début;
    const milliseconds = durée % 1000;
    const seconds = Math.floor(durée / 1000);
    // formatage durée
```

```
85.      result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds, milliseconds);
86.    }
87.    // résultat
88.    return result;
89. }
```

Commentaires

- lignes 16-31 : on définit une fonction **[async01]** qui rend son résultat au bout d'1 seconde via un événement de timer. La fonction **[async01]** reçoit un paramètre qui est utilisé dans son résultat ligne 27 ;
- lignes 35-50 : on fait de même avec une fonction **[async02]** qui rend son résultat au bout de 2 secondes via un événement de timer. La fonction **[async02]** admet deux paramètres qui sont utilisés dans son résultat ligne 46 ;
- lorsqu'elles seront appelées les deux fonctions **[async01, async02]** :
  - seront lancées ;
  - rendront au code appelant deux promesses **[promise1, promise2]** ;
  - l'exécution reviendra alors au code appelant qui continuera sa course ;
  - au bout d'1 seconde environ **[async01]** émettra un événement pour dire qu'elle a terminé son travail. L'événement en question sera mis en attente dans la boucle événementielle associé au résultat transmis par **[async01]** avec l'événement ;
  - au bout de 2 secondes environ, le même processus se passera pour **[async02]** ;
- ligne 56 : ce n'est que maintenant que les fonctions asynchrones **[async01, async02]** sont exécutées (notations async01(10) et async02(10,20)). Elles le sont au sein d'un tableau passé en paramètre à la méthode **[Promise.all]**. On sait que **[async01, async02]** rendent toutes deux une promesse au code appelant. Aussi le paramètre de **[Promise.all]** est un tableau de deux promesses ;
- **[Promise.all([promise1, promise2, …, promisen]).then(f1).catch(f2).finally(f3)**] est un abonnement à des événements :
  - **[Promise.all]** est de type **[Promise]** ;
  - la fonction **[f1]** de la méthode **[then]** sera exécutée lorsque **toutes les promesses [promise1, promise2, …, promisen]** du tableau paramètre de la méthode **[all]** seront passées de l'état **[pending]** à l'état **[resolved]**. Dit autrement, **[f1]** sera exécutée lorsque toutes les promesses du tableau se seront terminées avec succès ;
  - la fonction **[f2]** de la méthode **[catch]** sera exécutée dès que l'une des promesses du tableau passera de l'état **[pending]** à l'état **[rejected]**. Dit autrement, **[f2]** est exécutée dès que l'une des promesses du tableau échoue ;
  - la fonction **[f3]** de la méthode **[finally]** sera exécutée après l'exécution d'une des méthodes **[then, catch]**, donc toujours exécutée ;

L'exécution du code donne les résultats suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\async\async-05.js"
2   [début du script], heure=12:17:17:367
3   [début de la tâche asynchrone async01]
4   [début de la tâche asynchrone async02]
5   [fin du code principal du script], heure=12:17:17:375, durée= 0 seconde(s) et 10 millisecondes
6   [fin de la tâche asynchrone async01], heure=12:17:18:391, durée= 1 seconde(s) et 17 millisecondes
7   [fin de la tâche asynchrone async02], heure=12:17:19:389, durée= 2 seconde(s) et 14 millisecondes
8   [promise-
    all success], heure=12:17:19:390, durée= 2 seconde(s) et 25 millisecondes, result=[{"prop1":[10,20,30],"pro
    p2":"abcd","prop3":10},{"prop1":[11,21,31],"prop2":"xyzt","prop3":30}]
9   [promise-all finally], heure=12:17:19:392, durée= 2 seconde(s) et 27 millisecondes
10
11  [Done] exited with code=0 in 2.572 seconds
```

- lignes 6-7 : les deux tâches asynchrones **[async01, async02]** sont lancées. Elles fonctionnent en parallèle. Ce n'est pas l'exécution de leur code qui est faite en parallèle mais leurs attentes respectives de la donnée demandée se passent en même temps ;
- ligne 5 : le code principal du script est terminé. Restent à attendre la fin des deux tâches asynchrones **[async01, async02]** ;
- ligne 6 : la tâche asynchrone **[async01]** se termine environ 1 s après son lancement. Elle rend un résultat avec la fonction **[resolve]** donc sa promesse dans le tableau de la ligne 56 du code, passe de l'état **[pending]** à **[resolved]**. Ce n'est pas suffisant pour déclencher la méthode **[then]**, lignes 59-60 du code ;
- ligne 7 : la tâche asynchrone **[async02]** se termine environ 2 s après son lancement. Elle rend un résultat avec la fonction **[resolve]** donc sa promesse dans le tableau de la ligne 56 du code, passe de l'état **[pending]** à **[resolved]**. La méthode **[then]** va être exécutée dès que la boucle événementielle le permettra ;
- ligne 8 : la méthode **[then]** de **[Promise.all]** est exécutée. Elle reçoit en paramètre un tableau **[result1, result2]** où **[result1]** est le résultat émis par **[async01]**, et **[result2]** celui émis par **[async02]** ;
- ligne 9 : la méthode **[finally]** de **[Promise.all]** est exécutée ;

2.11.6 script [async-06]

Ce nouveau script montre comment l'utilisation conjointe des mots clés **[async / await]** permet d'avoir un code **asynchrone** ressemblant à un code **synchrone**. La gestion des événements est complètement cachée et la compréhension du code facilitée.

Nous reprenons l'exemple précédent en y amenant les modification suivantes :

- on ajoute une troisième fonction asynchrone **[async03]** qui elle renvoie son résultat avec la méthode **[Promise.reject]** qui signale donc à la boucle événementielle qu'elle a « échoué » à faire son travail ;
- on exécute séquentiellement les trois fonctions asynchrones **[async01, async02, async03]**. Dans l'exemple précédent, on avait exécuté en parallèle les fonctions asynchrones **[async01, async02]** ;
- avant l'apparition des mots clés **[async/await]**, l'exécution séquentielle d'actions asynchrones se faisait à l'aide de **[Promise]** emboîtées les unes dans les autres. Dès qu'il y avait plusieurs actions asynchrones à exécuter ainsi, le nombre de promesses augmentait en conséquence et le code devenait moins lisible ;
- avec les mots clés **[async/await]**, l'exécution séquentielle de tâches asynchrones se fait avec une syntaxe analogue à celle de l'exécution de tâches synchrones :

```
1   // fonction asynchrone - utilisation async / await
2   async function main() {
3     // exécution séquentielle des tâches asynchrones
4     try {
5       // exécution avec attente de [async01]
6       const result1 = await async01(...);
7       console.log("[async01 result]=", result1);
8       // exécution avec attente de [async02]
9       const result2 = await async02(...);
10      console.log("[async02 result]=", result2);
11      // exécution avec attente de [async03]
12      const result3 = await async03(...);
13      console.log("[async03 result]=", result3);
14    } catch (error) {
15      // une des actions asynchrones a échoué
16      console.log(sprintf("[sequential error]= %j, %s", error));
17    } finally {
18      // terminé
19      console.log("[fin exécution séquentielle des tâches asynchrones],");
20    }
```

- ligne 6 : la fonction asynchrone **[async01]** est lancée (mot clé **await**) et on attend qu'elle ait publié son résultat par l'une des méthodes **[Promise.resolve, Promise.reject]**. C'est donc une opération **bloquante** ;
- ligne 6 : le mot clé **[await]** transforme l'opération asynchrone **[async01]** en opération bloquante. On sait que l'opération **[async01]** rend un résultat de deux façons :
  - elle **rend** au code appelant, quasi immédiatement, un objet **[Promise]** ;
  - elle **publie** ultérieurement un résultat sur la boucle événementielle via les méthodes **[Promise.resolve, Promise.reject]**. C'est ce dernier résultat que récupère **[result1]**, ligne 6. La gestion événementielle de l'action **[async01]** est devenue invisible ;
  - si le résultat **[result]** de **[async01]** est publié par **[Promise.resolve(result)]**, il est affecté à **[result1]** ligne 6 et l'exécution continue ligne 7 ;
  - si le résultat de **[async01]** est publié par **[Promise.reject]**, cela provoque une exeption et l'exécution du code passe à la ligne 14, celle du **catch**. Le paramètre de la clause **[catch]** est l'objet d'erreur (error) publié par **[async01]** avec une expression **[Promise.reject(error)]**. La tâche asynchrone peut également publier l'erreur par un **[throw(error)]**. L'objet **[error]** est celui récupéré dans **[catch(error)]** ;
  - le mot clé **[await]** doit être obligatoirement dans une fonction précédée du mot clé **[async]**, ligne 2. Ce mot clé indique que la fonction **[main]** est une fonction asynchrone ;
  - dans l'expression **[await f(…)]**, **[f]** doit être une fonction asynchrone rendant un objet **[Promise]** au code appelant ;
- on refait la même chose pour l'action asynchrone **[async02]**, ligne 9 et **[async03]**, ligne 12 ;

Toujours avec les mots clés **[async / await]**, il est possible de faire l'exécution parallèle de tâches asynchrones avec la syntaxe suivante :

```
1   try {
2       // exécution parallèle des tâches asynchrones
3       const result = await Promise.all([async01(...), async02(...), async03(...)]);
4       console.log(sprintf("[parallel success], %s, result=%j", heure(débutParallel), result));
5     } catch (error) {
6       // une des actions asynchrones a échoué
7       console.log(sprintf("[parallel error], %s, erreur=%j", heure(débutParallel), error));
8     } finally {
9       // terminé
10      console.log(sprintf("[fin exécution parallèle des tâches asynchrones],%s", heure(débutParallel)));
11  }
```

- ligne 3 : on a une opération **bloquante** : on attend que les trois tâches asynchrones du tableau **[async01(..), async02(..), async03(..)]** aient publié leurs résultats sur la boucle événementielle avec l'une des méthodes **[Promise.resolve, Promise.reject]** ;
- si les trois tâches asynchrones publient leurs résultats avec **[Promise.resolve]**, la constante **[result]** est alors le tableau **[result1, result2, result3]** où :
  - **[result1]** est le résultat publié par **[async01]** avec l'expression **[Promise.resolve(result1)]** ;
  - **[result2]** est le résultat publié par **[async02]** avec l'expression **[Promise.resolve(result2)]** ;
  - **[result3]** est le résultat publié par **[async03]** avec l'expression **[Promise.resolve(result3)]** ;
- si l'une des trois tâches publie son résultat avec une expression **[Promise.reject(error)]** alors une exception se produit ;
  - la constante **[result]** de la ligne 3 ne reçoit pas sa valeur ;
  - l'exécution passe directement au **[catch]** de la ligne 5 ;
  - le paramètre (error) du catch, est l'objet (error) publié par l'expression **[Promise.reject(error)]** ;

En mixant ces deux syntaxes, on peut exécuter indifféremment des tâches asynchrones en séquentiel ou en parallèle, tout cela avec une syntaxe analogue à celle d'un code synchrone. Il faut donc privilégier cette syntaxe beaucoup plus lisible que les précédentes. Cette syntaxe **[async / await]** n'est disponible que depuis la version 6 d'ECMAScript. Il y a encore beaucoup de codes Javascript utilisant des promesses **[Promise]**. C'est pourquoi il est important de comprendre également le fonctionnement de celles-ci.

Le code complet du script **[async-06]** est le suivant :

```
1.  'use strict';
2.
3.  // exécution parallèle ou séquentielle de plusieurs tâches asynchrones
4.  // avec les mots clés async / await
5.
6.  // imports
7.  import moment from 'moment';
8.  import { sprintf } from 'sprintf-js';
9.
10. // début
11. const débutScript = moment(Date.now());
12. console.log("[début du code principal du script],", heure());
13.
14. // une fonction asynchrone rendant une [Promise]
15. function async01(débutAsync01) {
16.   return new Promise(function (resolve) {
17.     console.log("[début fonction asynchrone async01],", heure());
18.     // fonction asynchrone
19.     setTimeout(function () {
20.       console.log("[fin fonction asynchrone async01],", heure(débutAsync01));
21.       // l'action asynchrone peut rendre un résultat complexe
22.       // ici réussite
23.       resolve({
24.         prop1: [11, 21, 31],
25.         prop2: "abcd"
26.       });
27.     }, 1000)
28.   });
29. }
30.
31. // une fonction asynchrone rendant une [Promise]
32. function async02(débutAsync02) {
33.   console.log("[début fonction asynchrone async02],", heure());
34.   return new Promise(function (resolve) {
35.     // fonction asynchrone
36.     setTimeout(function () {
37.       console.log("[fin fonction asynchrone async02],", heure(débutAsync02));
38.       // l'action asynchrone peut rendre un résultat complexe
39.       // ici réussite
40.       resolve({
41.         prop1: [12, 22, 32],
42.         prop2: "xyzt"
43.       });
44.     }, 2000)
45.   })
46. }
47.
48. // une fonction asynchrone rendant une [Promise]
49. function async03(débutAsync03) {
50.   console.log("[début fonction asynchrone async03],", heure());
51.   return new Promise((resolve, reject) => {
52.     // fonction asynchrone
53.     setTimeout(function () {
54.       console.log("[fin fonction asynchrone async03],", heure(débutAsync03));
55.       // l'action asynchrone peut rendre un résultat complexe
56.       // ici échec
57.       reject({
58.         prop1: [13, 23, 33],
59.         prop2: "échec"
```

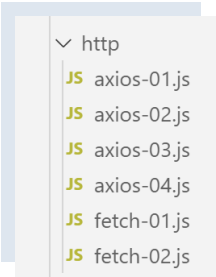
```
60.       });
61.     }, 3000)
62.   })
63. }
64.
65. // fonction asynchrone - utilisation async / await
66. async function main() {
67.   const débutSequential = moment(Date.now());
68.   // exécution séquentielle des tâches asynchrones
69.   console.log("------------ exécution séquentielle des tâches asynchrones lancée ------------------------")
70.   try {
71.     // exécution avec attente de [async01]
72.     const débutAsync01 = moment(Date.now());
73.     const result1 = await async01(débutAsync01);
74.     console.log("[async01 result]=", result1);
75.     // exécution avec attente de [async02]
76.     const débutAsync02 = moment(Date.now());
77.     console.log("début async02-------------", heure());
78.     const result2 = await async02(débutAsync02);
79.     console.log("[async02 result]=", result2);
80.     // exécution avec attente de [async03]
81.     const débutAsync03 = moment(Date.now());
82.     console.log("début async03-------------", heure());
83.     const result3 = await async03(débutAsync03);
84.     console.log("[async03 result]=", result3);
85.   } catch (error) {
86.     // une des actions asynchrones a échoué
87.     console.log(sprintf("[sequential error]= %j, %s", error, heure(débutSequential)));
88.   } finally {
89.     // terminé
90.     console.log("[fin exécution séquentielle des tâches asynchrones],", heure(débutSequential));
91.   }
92.
93.   const débutParallel = moment(Date.now());
94.   // exécution en parallèle des tâches asynchrones
95.   console.log("------------ exécution parallèle des tâches asynchrones lancée ------------------------");
96.   try {
97.     const result = await Promise.all([async01(débutParallel), async02(débutParallel), async03(débutParallel)]);
98.     console.log(sprintf("[parallel success], %s, result=%j", heure(débutParallel), result));
99.   } catch (error) {
100.    // une des actions asynchrones a échoué
101.    console.log(sprintf("[parallel error], %s, erreur=%j", heure(débutParallel), error));
102.  } finally {
103.    // terminé
104.    console.log(sprintf("[fin exécution parallèle des tâches asynchrones],%s", heure(débutParallel)));
105.  }
106.
107.  // terminé
108.  console.log("[fin de la fonction main],", heure(débutSequential));
109. }
110. // exécution fonction asynchrone main
111. main();
112.
113. // s'affichera avant les différents msgs des fonctions asynchrones et de leurs évts
114. console.log("[fin du code principal du script],", heure(débutScript));
115.
116. // utilitaire
117. function heure(début) {
118.   // heure du moment courant
119.   const now = moment(Date.now());
120.   // formatage heure
121.   let result = "heure=" + now.format("HH:mm:ss:SSS");
122.   if (début) {
123.     const durée = now - début;
124.     const milliseconds = durée % 1000;
125.     const seconds = Math.floor(durée / 1000);
126.     // formatage durée
127.     result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds, milliseconds);
128.   }
129.   // résultat
130.   return result;
131. }
```

Les résultats de l'exécution sont les suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\async\async-06.js"
2   [début du code principal du script], heure=15:02:00:152
3   ------------ exécution séquentielle des tâches asynchrones lancée ------------------------
4   [début fonction asynchrone async01], heure=15:02:00:161
5   [fin du code principal du script], heure=15:02:00:164, durée= 0 seconde(s) et 15 millisecondes
6   [fin fonction asynchrone async01], heure=15:02:01:165, durée= 1 seconde(s) et 4 millisecondes
7   [async01 result]= { prop1: [ 11, 21, 31 ], prop2: 'abcd' }
8   début async02------------- heure=15:02:01:253
```

```
9   [début fonction asynchrone async02], heure=15:02:01:254
10  [fin fonction asynchrone async02], heure=15:02:03:265, durée= 2 seconde(s) et 12 millisecondes
11  [async02 result]= { prop1: [ 12, 22, 32 ], prop2: 'xyzt' }
12  début async03------------ heure=15:02:03:268
13  [début fonction asynchrone async03], heure=15:02:03:268
14  [fin fonction asynchrone async03], heure=15:02:06:285, durée= 3 seconde(s) et 18 millisecondes
15  [sequential error]= {"prop1":[13,23,33],"prop2":"échec"}, heure=15:02:06:289, durée= 6 seconde(s) et 129 mi
    llisecondes
16  [fin exécution séquentielle des tâches asynchrones], heure=15:02:06:291, durée= 6 seconde(s) et 131 millise
    condes
17  ------------ exécution parallèle des tâches asynchrones lancée ------------------------
18  [début fonction asynchrone async01], heure=15:02:06:292
19  [début fonction asynchrone async02], heure=15:02:06:293
20  [début fonction asynchrone async03], heure=15:02:06:294
21  [fin fonction asynchrone async01], heure=15:02:07:294, durée= 1 seconde(s) et 2 millisecondes
22  [fin fonction asynchrone async02], heure=15:02:08:298, durée= 2 seconde(s) et 6 millisecondes
23  [fin fonction asynchrone async03], heure=15:02:09:297, durée= 3 seconde(s) et 5 millisecondes
24  [parallel error], heure=15:02:09:298, durée= 3 seconde(s) et 6 millisecondes, erreur={"prop1":[13,23,33],"p
    rop2":"échec"}
25  [fin exécution parallèle des tâches asynchrones],heure=15:02:09:299, durée= 3 seconde(s) et 7 millisecondes
26  [fin de la fonction main], heure=15:02:09:300, durée= 9 seconde(s) et 140 millisecondes
27
28  [Done] exited with code=0 in 9.668 seconds
```

2.12 Les fonctions HTTP de Javascript



2.12.1 Choix d'une bibliothèque HTTP

Nous avons fait ici le choix de deux bibliothèques :

EcmaScript 6 a nativement une fonction HTTP appelée **[fetch]** qui n'est pas implémentée par **[node.js]** (sept 2019). Il existe une bibliothèque appelée **[node-fetch]** qui permet d'utiliser la fonction **[fetch]** sous Node. Cette bibliothèque utilise certaines API propres à **[node.js]**. Un code **[node-fetch]** peut être alors non transportable à 100 % dans un environnement non **[node]**, dans un navigateur par exemple ;

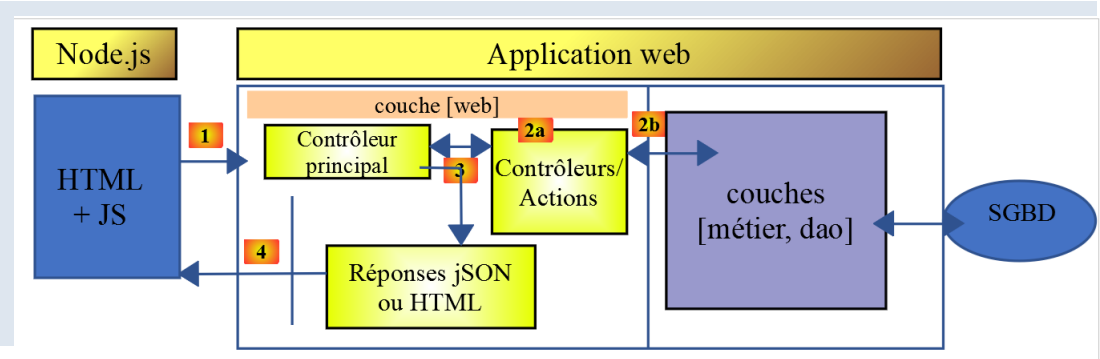
Il existe par ailleurs une bibliothèque nommée **[axios]** dédiée aux requêtes HTTP compatible aussi bien avec **[node.js]** qu'avec les navigateurs. C'est cette bibliothèque que nous utiliserons au final.

Nous allons présenter un même script écrit avec ces deux bibliothèques pour montrer que la démarche de codage avec elles est analogue.

2.12.2 Mise en place d'un environnement de travail

2.12.2.1 Installation du serveur de calcul d'impôt

Ultimement, nous allons écrire une application web avec l'architecture suivante :



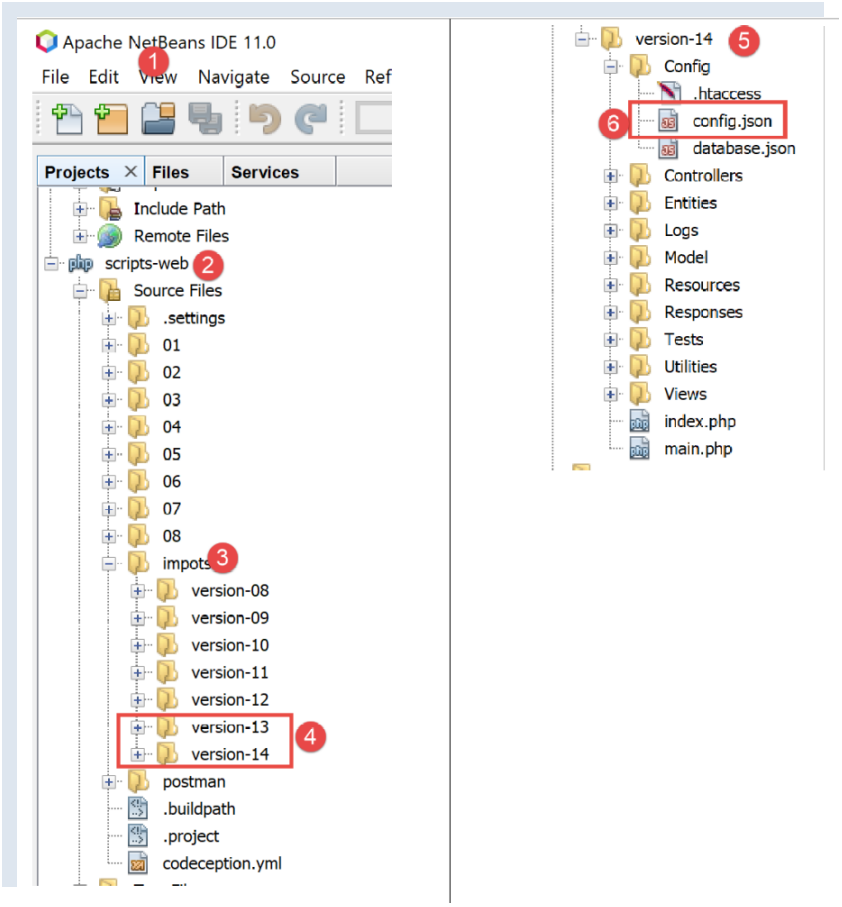
JS : Javascript

Le code Javascript est client :

- d'un service de pages ou fragments statiques ;
- d'un service jSON ;

Le code Javascript est donc un client jSON et à ce titre peut être organisé en couches **`[UI, métier, dao]`** (UI : User Interface) comme l'ont été nos clients jSON écrits en PHP.

Le serveur sera celui du calcul de l'impôt dont nous avons déjà écrit 13 versions. Nous allons en écrire une 14ième. Nous commençons donc par dupliquer, sous Netbeans, le dossier de la version 13, dans le dossier de la version 14 :



- en **`[6]`**, nous modifions le fichier **`[config.json]`** de la version 14 de la façon suivante :

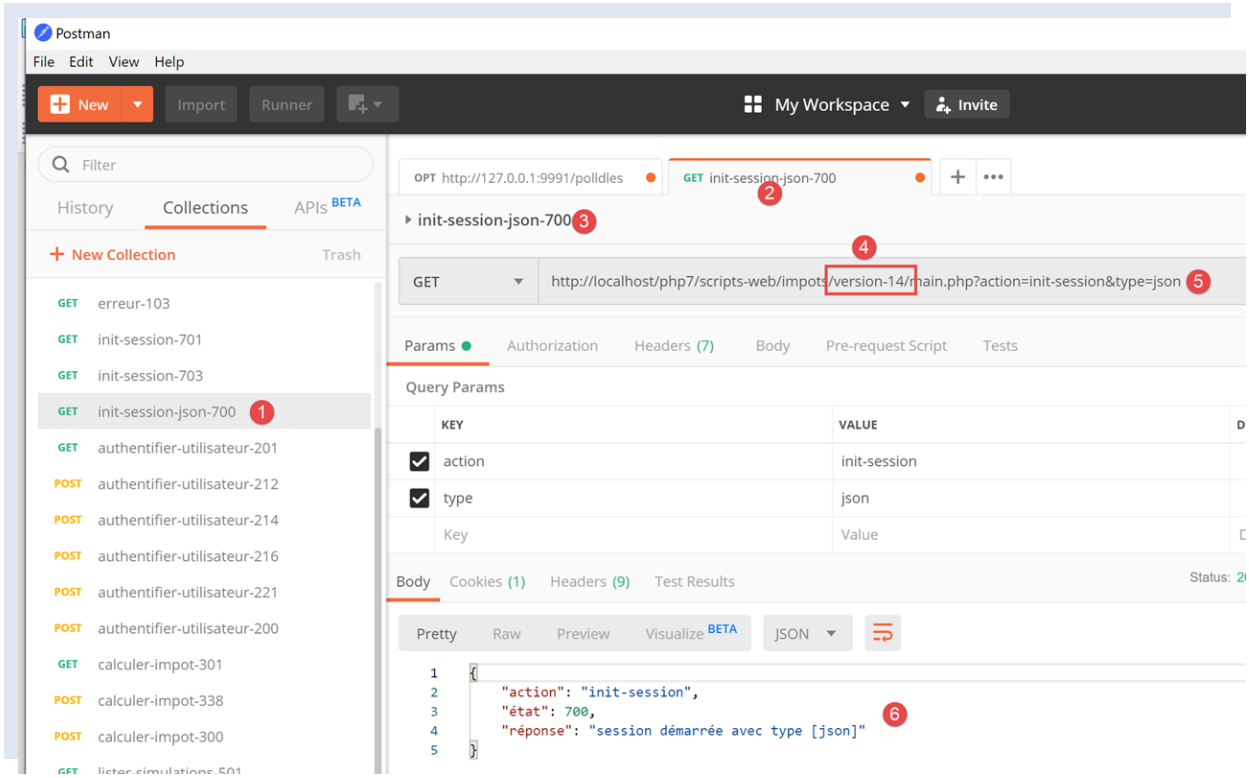
```
{
    "databaseFilename": "Config/database.json",
    "rootDirectory": "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-14",
    "relativeDependencies": [
```

```
5
6           "/Entities/BaseEntity.php",
7           "/Entities/Simulation.php",
8           ...
9       "vues": {
10          "vue-authentification.php": [700, 221, 400],
11          "vue-calcul-impot.php": [200, 300, 341, 350, 800],
12          "vue-liste-simulations.php": [500, 600]
13      },
14      "vue-erreurs": "vue-erreurs.php"
15  }
```

- ligne 3, nous changeons le dossier racine de l'application ;

Pour accéder à ce serveur, il faut lancer les services **[Laragon]**.

Ceci fait, nous pouvons tester avec **[Postman]** (cf. article lien), cette nouvelle version du serveur, identique pour l'instant à la version 13. Nous pouvons utiliser la collection de requêtes utilisées pour tester la version 12 du serveur de calcul d'impôt :



- en **[1-4]**, utiliser la requête **[init-session-700]** pour initialiser une session jSON ;
- en **[4-5]**, mettre **[version-14]** au lieu de **[version-12]** pour tester la version 14 du projet ;
- à l'exécution on doit recevoir la réponse jS0N **[6]** du serveur ;

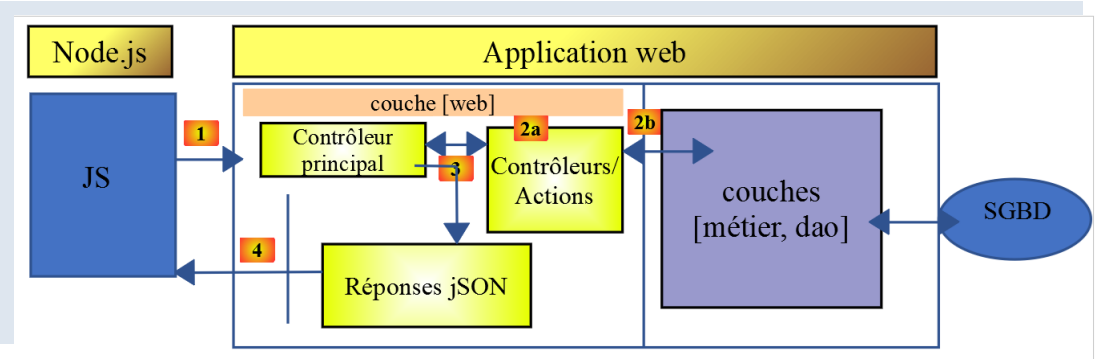
La version 14 du serveur est désormais opérationnelle. Nous serons amenés à la modifier légèrement. Rappelons l'API de ce serveur :

Action	Rôle	Contexte d'exécution
init-session	Sert à fixer le type (json, xml, html) des réponses souhaitées	Requête **GET main.php?action=init-session&type=x** peut être émise à tout moment
authentifier-utilisateur	Autorise ou non un utilisateur à se connecter	Requête **POST main.php?action=authentifier-utilisateur** La requête doit avoir deux paramètres postés **[user, password]** Ne peut être émise que si le type de la session (json, xml, html) est connu
calculer-impot	Fait une simulation de calcul d'impôt	Requête **POST main.php?action=calculer-impot** La requête doit avoir trois paramètres postés **[marié, enfants, salaire]** Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
lister-simulations	Demande à voir la liste des simulations opérées depuis le début de la session	Requête **GET main.php?action=lister-simulations** La requête n'accepte aucun autre paramètre

		Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
supprimer-simulation	Supprime une simulation de la liste des simulations	Requête **GET main.php?action=lister-simulations&numéro=x** La requête n'accepte aucun autre paramètre Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié
fin-session	Termine la session de simulations.	Techniquement l'ancienne session web est supprimée et une nouvelle session est créée Ne peut être émise que si le type de la session (json, xml, html) est connu et l'utilisateur authentifié

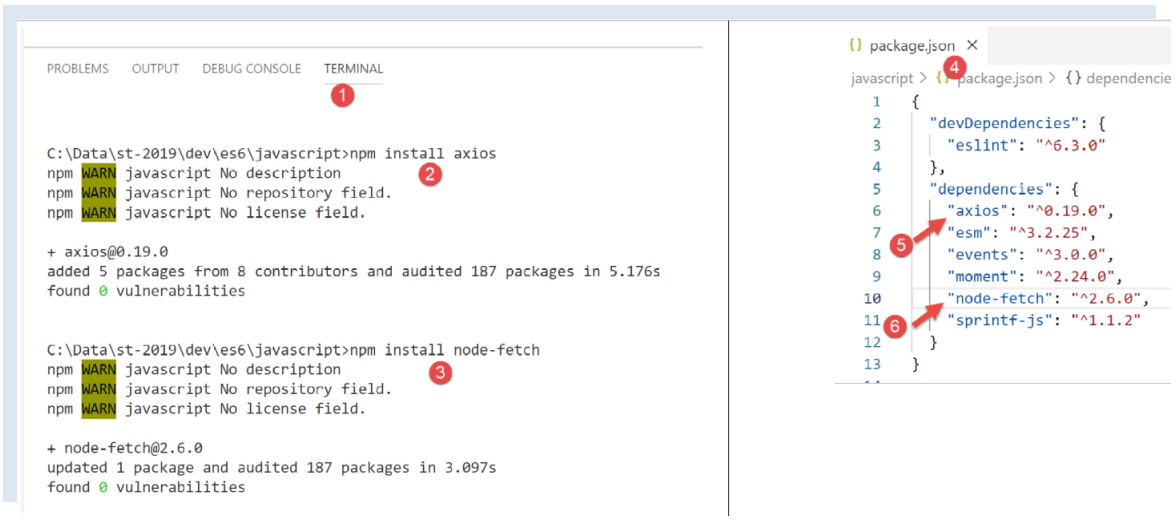
2.12.2.2 Installation des bibliothèques HTTP du client Javascript

Dans un premier temps, nous travaillerons avec l'architecture suivante :

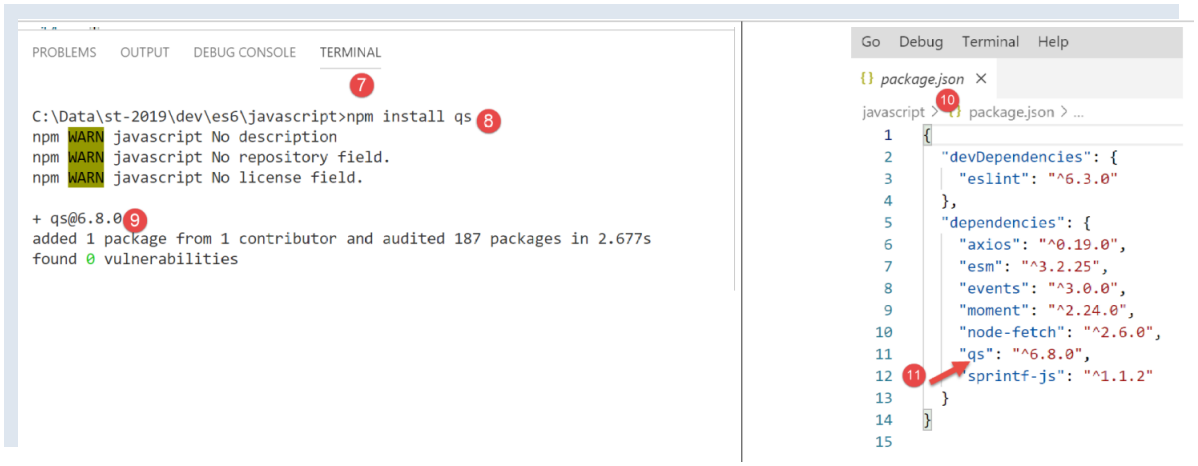


- en **[1]**, un script console **[node.js]** fait une requête HTTP vers le serveur jSON du calcul de l'impôt ;
- en **[4]**, il reçoit cette réponse et l'affiche sur la console ;

Dans l'exemple n° 1, nous utiliserons les bibliothèques **[node-fetch]** et **[axios]** puis nous ne conserverons qu'**[axios]** pour les exemples suivants. Nous installons maintenant ces deux bibliothèques Javascript à partir du terminal de **[VSCode]** :



Nous utiliserons également la bibliothèque **[qs]** qui permet l'encodage URL d'une chaîne de caractères. On se rappelle que cet encodage est utilisé pour encoder les paramètres d'une requête HTTP GET ou POST.



2.12.3 script `[fetch-01]`

Le script **`[fetch-01]`** utilise la bibliothèque **`[node-fetch]`** pour initialiser une session jSON avec le serveur de calcul d'impôt. Son code est le suivant :

```
1.  'use strict';
2.
3.  // imports
4.  import fetch from 'node-fetch';
5.  import qs from 'qs';
6.  import { sprintf } from 'sprintf-js';
7.  import moment from 'moment';
8.
9.
10. // URL de base du serveur de calcul d'impôt
11. const baseUrl = 'http://localhost/php7/scripts-web/impots/version-14/main.php?';
12. // init session
13. async function initSession() {
14.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
15.   const options = {
16.     method: "GET",
17.     timeout: 2000
18.   };
19.   // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
20.   let débutFetch;
21.   try {
22.     // requête asynchrone - [fetch] rend une promesse
23.     débutFetch = moment(Date.now());
24.     const response = await fetch(baseUrl + qs.stringify({
25.       action: 'init-session',
26.       type: 'x'
27.     }), options);
28.     // [response] est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
29.     // on affiche cette réponse pour voir sa structure
30.     console.log(sprintf("réponse fetch formatée en json,=%j, %s", response, heure(débutFetch)));
31.     console.log("réponse fetch en javascript=", response);
32.     // on peut avoir aux entêtes HTTP
33.     console.log("entêtes de la réponse=", response.headers);
34.     // si réponse de type application / json, la réponse json du serveur est obtenue avec la fonction asynchrone
    [response.json()]
35.     // dans ce cas le code appelant obtient un objet [Promise]
36.     // [await] permet d'obtenir la réponse [json] du serveur plutôt que sa promesse
37.     const débutJson = moment(Date.now());
38.     const objet = await response.json();
39.     console.log(sprintf("réponse json=%j, type=%s, %s", objet, typeof (objet), heure(débutJson)));
40.     return objet;
41.     // si réponse de type text / plain, la réponse texte du serveur est obtenue avec [response.text()]
42.     // dans ce cas le code appelant obtient un objet [Promise]
43.     // [await] permet d'obtenir la réponse [texte] du serveur plutôt que sa promesse
44.     // const text = await response.text();
45.     // console.log("réponse texte=", text);
46.     // return text;
47.   } catch (error) {
48.     // on est là parce que le serveur a envoyé un code d'erreur [404 Not Found, ...] accompagné d'un corps vide - on
    affiche l'erreur pour voir sa structure
49.     // ou bien parce que le client [fetch] a lancé une exception (réseau inaccesible, ...)
50.     // on affiche la structure de l'erreur
51.     console.log(sprintf("error fetch en json=%j, %s", error, heure(débutFetch)));
52.     console.log("error fetch en javascript=", typeof (error), error);
53.     // on lance le msg d'erreur reçu
54.     throw error.message;
```

```
55.     }
56.   }
57.
58.   // la fonction main exécute la fonction asynchrone [initSession]
59.   async function main() {
60.     try {
61.       console.log("requête HTTP vers le serveur en cours ----------------------------------------------");
62.       const response = await initSession();
63.       console.log("succès -----------------------------------------------");
64.       console.log("réponse=", response, typeof (response))
65.     } catch (error) {
66.       console.log("erreur -----------------------------------------------");
67.       console.log("erreur=", error, typeof (error));
68.     }
69.   }
70.
71.   // test
72.   main();
73.
74.   // utilitaire d'affichage heure et durée
75.   function heure(début) {
76.     // heure du moment courant
77.     const now = moment(Date.now());
78.     // formatage heure
79.     let result = "heure=" + now.format("HH:mm:ss:SSS");
80.     // faut-il calculer une durée ?
81.     if (début) {
82.       const durée = now - début;
83.       const milliseconds = durée % 1000;
84.       const seconds = Math.floor(durée / 1000);
85.       // formatage heure + durée
86.       result = result + sprintf(", durée= %s seconde(s) et %s millisecondes", seconds, milliseconds);
87.     }
88.     // résultat
89.     return result;
90.   }
```

Commentaires

- les fonctions HTTP du Javascript sont des fonctions asynchrones. Nous utilisons ici ce que nous avons appris dans la section précédente (cf. lien) ;
- ligne 24 : pour attendre que la réponse de la fonction asynchrone **[fetch]** soit publiée sur la boucle événementielle de **[node.js]**, nous utilisons le mot clé **[await]**. Nous savons qu'alors que cette instruction doit être dans un code préfixé par le mot clé **[async]** (ligne 13) ;
- lignes 13-56 : nous encapsulons le code HTTP dans la fonction asynchrone **[initSession]** ;
- lignes 59-69 : une seconde fonction asynchrone **[main]** est utilisée pour appeler de façon bloquante (async / await) la fonction asynchrone **[initSession]** ;
- ligne 72 : la fonction asynchrone **[main]** est appelée ;
- bien que l'ensemble du code ressemble à du code synchrone, ce sont bien des fonctions asynchrones qui sont exécutées, mais de façon bloquante ;
- ligne 19 : pour initialiser une session jSON avec le serveur de calcul d'impôt, il faut lui envoyer la commande HTTP **[get /main.php?action=init-session&type=json]**. C'est ce que fait le code des lignes 24-27. La syntaxe de **[fetch]** est la suivante **[fetch(URL, options)]** avec :
  - **[URL]** : l'URL interrogée ;
  - **[options]** : un objet définissant les options de la requête. C'est là notamment qu'on définit les entêtes HTTP qu'on veut envoyer à la machine cible ;
- lignes 15-18 : on définit les options de la requête qu'on veut faire :
  - **[method]** : on veut faire un GET ;
  - **[timeout]** : on veut que le client **[fetch]** n'attende pas plus de 2 secondes la réponse du serveur Si ce délai est dépassé, **[fetch]** lancera une exception ;
- ligne 24 : pour obtenir l'URL **[/main.php?action=init-session&type=json]**, on utilise la bibliohèque **[qs]** pour obtenir l'encodage URL des paramètres **[action,type]** du GET. La chaîne obtenue est **[init-session&type=json]** qu'on aurait pu construire nous-mêmes. On voulait simplement montrer comment obtenir une chaîne URL encodée ;
- ligne 24 : le mot clé **[await]** montre que c'est une tâche asynchrone qui est lancée ici et qu'on attend qu'elle publie sa réponse sur la boucle événementielle de **[node.js]** ;
- ligne 24 : dans **[response]**, on obtient un objet complexe qui décrit la totalité de la réponse HTTP reçue (entêtes et document) ;
- lignes 30-31 : on affiche l'objet **[response]** pour voir sa structure, d'abord comme chaîne de caractères puis comme objet Javascript ;
- ligne 33 : on affiche les entêtes HTTP envoyés par le serveur ;
- ligne 38 : on sait que le serveur de calcul d'impôt va envoyer une chaîne jSON. Celle-ci est encapsulée dans l'objet **[response]**. On peut l'obtenir avec la méthode **[response.json()]**. Cependant cette méthode est asynchrone. On écrit donc **[await response.json()]** pour obtenir la chaîne jSON qui va être publiée sur la boucle événementielle de **[node.js]**. En fait ce n'est pas la chaîne jSON qu'on obtient mais l'objet Javascript représenté par celle-ci ;
- ligne 39 : affichage de la chaîne jSON reçue ;

- ligne 40 : on rend l'objet Javascript reçu ;
- ligne 47 : on intercepte une erreur éventuelle de l'instruction **[fetch]**. Celle-ci ne lance une exception que si l'opération HTTP n'a pu aboutir et qu'aucune réponse du serveur n'a été reçue. Si une réponse a été reçue, même avec un code HTTP différent de **[200 OK]**, **[fetch]** ne lance pas d'exception et la réponse du serveur sera disponible ligne 38 ;
- lignes 51-52 : on affiche l'objet **[error]** reçu par la clause **[catch]**, d'abord comme une chaîne jSON puis comme un objet Javascript ;
- ligne 54 : le message d'erreur de **[fetch]** se trouve dans **[error.message]** ;
- lignes 59-69 : la fonction asynchrone **[main]** lance la fonction asynchrone **[initSession]** de façon bloquante (await ligne 62) ;
- ligne 72 : la fonction asynchrone **[main]** est lancée et le code principal du script est alors terminé. Le script global lui sera terminé lorsque les tâches asynchrones lancées auront publié leurs résultats sur la boucle événementielle ;

Les résultats de l'exécution sont les suivants :

Cas 1 : le serveur Laragon n'est pas lancé

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\http\fetch-01.js"
2   requête HTTP vers le serveur en cours ---------------------------------------------
3   error fetch en json={"message":"network timeout at: http://localhost/php7/scripts-web/impots/version-
    14/main.php?action=init-session&type=json","type":"request-
    timeout"}, heure=10:08:48:180, durée= 2 seconde(s) et 62 millisecondes
4   error fetch en javascript= object { FetchError: network timeout at: http://localhost/php7/scripts-
    web/impots/version-14/main.php?action=init-session&type=json
5       at Timeout.<anonymous> (c:\Data\st-2019\dev\es6\javascript\node_modules\node-
    fetch\lib\index.js:1448:13)
6       at ontimeout (timers.js:436:11)
7       at tryOnTimeout (timers.js:300:5)
8       at listOnTimeout (timers.js:263:5)
9       at Timer.processTimers (timers.js:223:10)
10    message:
11     'network timeout at: http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=json',
12    type: 'request-timeout' }
13  erreur ----------------------------------------------
14  erreur= network timeout at: http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=json string
15
16  [Done] exited with code=0 in 2.804 seconds
```

Commentaires

- ligne 3 : la requête HTTP échoue au bout de 2 secondes et 62 millisecondes à cause du timeout de 2 secondes qu'on avait imposé à la requête HTTP ;
- lignes 4-9 : l'objet Javascript **[error]** intercepté par la clause **[catch(error)]**. Cet objet a deux propriétés :
  - **[FetchError]** : ligne 4 ;
  - **[message]** : lignes 10-12 ;
- ligne 14 : le message d'erreur reçu par la fonction asynchrone **[main]** ;

Cas 2 : le serveur Laragon est lancé

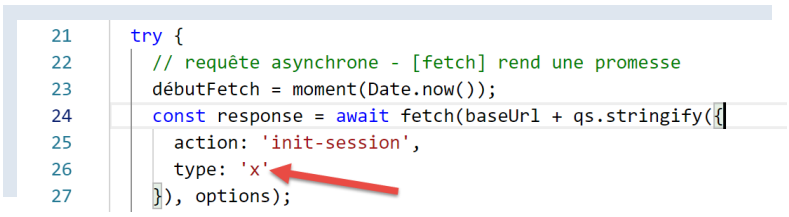
```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\http\fetch-01.js"
2   requête HTTP vers le serveur en cours ---------------------------------------------
3   réponse fetch formatée en json,={"size":0,"timeout":2000}, heure=10:13:50:814, durée= 0 seconde(s) et 375 m
    illisecondes
4   réponse fetch en javascript= Response {
5     size: 0,
6     timeout: 2000,
7     [Symbol(Body internals)]:
8      { body:
9         PassThrough {
10          _readableState: [ReadableState],
11          readable: true,
12          domain: null,
13          _events: [Object],
14          _eventsCount: 2,
15          _maxListeners: undefined,
16          _writableState: [WritableState],
17          writable: false,
18          allowHalfOpen: true,
19          _transformState: [Object] },
20       disturbed: false,
```

```
21        error: null },
22     [Symbol(Response internals)]:
23      { url:
24         'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json',
25        status: 200,
26        statusText: 'OK',
27        headers: Headers { [Symbol(map)]: [Object] },
28        counter: 0 } }
29  entêtes de la réponse= Headers {
30    [Symbol(map)]:
31     [Object: null prototype] {
32       date: [ 'Sat, 14 Sep 2019 08:13:50 GMT' ],
33       server: [ 'Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11' ],
34       'x-powered-by': [ 'PHP/7.2.11' ],
35       'cache-control': [ 'max-age=0, private, must-revalidate, no-cache, private' ],
36       'set-cookie': [ 'PHPSESSID=99q2iinusmhl55fa600aie2mmu; path=/' ],
37       'content-length': [ '86' ],
38       connection: [ 'close' ],
39       'content-type': [ 'application/json' ] } }
40  réponse json={"action":"init-
    session","état":700,"réponse":"session démarrée avec type [json]"}, type=object, heure=10:13:50:825, durée=
     0 seconde(s) et 1 millisecondes
41  succès ------------------------------------------
42  réponse= { action: 'init-session',
43    'état': 700,
44    'réponse': 'session démarrée avec type [json]' } object
45
46  [Done] exited with code=0 in 1.022 seconds
```

Commentaires

- ligne 3 : **[fetch]** reçoit la réponse du serveur au bout de 375 ms ;
- lignes 4-39 : la structure de l'objet Javascript **[response]** encapsulant la réponse du serveur. Parmi ses propriétés, certaines peuvent nous intéresser :
  - **[status]** (ligne 25) : code HTTP de la réponse du serveur ;
  - **[statusText]** (ligne 26) : texte associé à ce code ;
  - **[headers]** (ligne 27) : les entêtes HTTP de la réponse du serveur ;
  - **[body]** (ligne 8) : représente le document envoyé par le serveur. L'instruction **[fetch]** offre des méthodes pour l'exploiter ;
- lignes 29-39 : les entêtes HTTP de la réponse du serveur ;
- ligne 40 : la fonction asynchrone **[response.json()]** a publié sa réponse au bout d'1 milliseconde ;
- lignes 42-44 : l'objet Javascript reçu par la fonction asynchrone **[main]** ;

Cas 3 : le serveur Laragon est lancé mais on lui envoie une commande erronée :



- ci-dessus, ligne 26, on passe un type erroné de session au serveur ;

Les résultats de l'exécution sont les suivants :

```
1   requête HTTP vers le serveur en cours ------------------------------------------
2   réponse fetch formatée en json,={"size":0,"timeout":2000}, heure=10:27:54:114, durée= 0 seconde(s) et 136 m
    illisecondes
3   réponse fetch en javascript= Response {
4     size: 0,
5     timeout: 2000,
6     [Symbol(Body internals)]:
7      { body:
8         PassThrough {
9           _readableState: [ReadableState],
10          readable: true,
11          domain: null,
12          _events: [Object],
13          _eventsCount: 2,
14          _maxListeners: undefined,
```

```
15          _writableState: [WritableState],
16          writable: false,
17          allowHalfOpen: true,
18          _transformState: [Object] },
19       disturbed: false,
20       error: null },
21    [Symbol(Response internals)]:
22     { url:
23        'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-session&type=x',
24       status: 400,
25       statusText: 'Bad Request',
26       headers: Headers { [Symbol(map)]: [Object] },
27       counter: 0 } }
28  entêtes de la réponse= Headers {
29    [Symbol(map)]:
30     [Object: null prototype] {
31       date: [ 'Sat, 14 Sep 2019 08:27:54 GMT' ],
32       server: [ 'Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11' ],
33       'x-powered-by': [ 'PHP/7.2.11' ],
34       'cache-control': [ 'max-age=0, private, must-revalidate, no-cache, private' ],
35       'set-cookie': [ 'PHPSESSID=5ku9gfok81ikj98hia0meeum57; path=/' ],
36       'content-length': [ '79' ],
37       connection: [ 'close' ],
38       'content-type': [ 'application/json' ] } }
39  réponse json={"action":"init-
    session","état":703,"réponse":"paramètre type=[x] invalide"}, type=object, heure=10:27:54:127, durée= 0 sec
    onde(s) et 2 millisecondes
40  succès ------------------------------------------
41  réponse= { action: 'init-session',
42    'état': 703,
43    'réponse': 'paramètre type=[x] invalide' } object
44
45  [Done] exited with code=0 in 0.712 seconds
```

- la réponse du serveur est reçue ligne 2 ;
- ligne 24 : on peut voir que le code HTTP de la réponse du serveur est 400, un code d'erreur. Néanmoins, **[fetch]** n'a pas lancé d'exception. Tant que **[fetch]** reçoit une réponse du serveur, il l'exploite et ne lance pas d'exception ;
- lignes 41-43 : la réponse obtenue par la fonction asynchrone **[main]** ;

2.12.4 script [fetch-02]

Le script suivant reprend le script **[fetch-01]** en le débarrassant de tous les détails inutiles :

```
1.  'use strict';
2.
3.  // imports
4.  import fetch from 'node-fetch';
5.  import qs from 'qs';
6.
7.  // URL de base du serveur de calcul d'impôt
8.  const baseUrl = 'http://localhost/php7/scripts-web/impots/version-14/main.php?';
9.  // init session
10. async function initSession() {
11.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
12.   const options = {
13.     method: "GET",
14.     timeout: 2000
15.   };
16.   // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
17.   const response = await fetch(baseUrl + qs.stringify({
18.     action: 'init-session',
19.     type: 'json'
20.   }), options);
21.   // résultat reçu en jSON
22.   return await response.json();
23. }
24.
25. // la fonction main exécute la fonction asynchrone [initSession]
26. async function main() {
27.   try {
28.     console.log("requête HTTP vers le serveur en cours ------------------------------------------");
29.     const response = await initSession();
30.     console.log("succès ------------------------------------------");
31.     console.log("réponse=", response)
32.   } catch (error) {
```

```
33.     console.log("erreur ------------------------------------------");
34.     console.log("erreur=", error.message);
35.   }
36. }
37.
38. // test
39. main();
```

Les résultats d'une exécution normale :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\http\fetch-02.js"
2   requête HTTP vers le serveur en cours ------------------------------------------
3   succès -------------------------------------------
4   réponse= { action: 'init-session',
5     'état': 700,
6     'réponse': 'session démarrée avec type [json]' }
7
8   [Done] exited with code=0 in 0.56 seconds
```

Les résultats d'une exécution avec exception (on arrête le serveur Laragon) :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\http\fetch-02.js"
2   requête HTTP vers le serveur en cours ------------------------------------------
3   erreur -------------------------------------------
4   erreur= network timeout at: http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=json
5
6   [Done] exited with code=0 in 2.701 seconds
```

2.12.5 script [axios-01]

On reprend ici le script **[fetch-01]** que l'on réécrit avec la bibliothèque **[axios]**. On rappelle que notre intérêt pour cette bibliothèque est qu'elle soit portable entre l'environnement **[node.js]** et ceux des navigateurs usuels. Cela permet :

- dans une phase 1, de tester nos scripts dans un environnement **[node.js]** ;
- dans une phase 2, de les porter sur un navigateur ;

Le script **[axios-01]** reprend la structure du script **[fetch-01]** :

```
1.  'use strict';
2.  import axios from 'axios';
3.
4.  // configuration par défaut d'axios
5.  axios.defaults.timeout = 2000;
6.  axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
7.
8.  // init session
9.  async function initSession(axios) {
10.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
11.   const options = {
12.     method: "GET",
13.     // paramètres de l'URL
14.     params: {
15.       action: 'init-session',
16.       type: 'x'
17.     }
18.   };
19.   // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
20.   try {
21.     // requête asynchrone
22.     const response = await axios.request('main.php', options);
23.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
24.     // on affiche cette réponse pour voir sa structure
25.     console.log("réponse axios=", response);
26.     // la réponse du serveur est dans [response.data]
27.     return response.data;
28.   } catch (error) {
29.     // on est là parce que le serveur a envoyé un code d'erreur [404 Not Found, 500 Internal Server Error, ...]
30.     // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
31.     // on l'affiche pour voir sa structure
32.     console.log("axios error=", typeof (error), error);
33.     if (error.response) {
34.       // le serveur a signalé une erreur dans le statut HTTP mais il a aussi envoyé une réponse
35.       // alors celle-ci est trouvée dans [error.response.data]
36.       // on sait que le serveur envoie des réponses jSON de structure {action, état, réponse}
```

```
37.       // et qu'en cas d'erreur, le msg d'erreur est dans [réponse]
38.       return error.response.data;
39.     } else {
40.       // on lance l'erreur
41.       throw error;
42.     }
43.   }
44. }
45.
46. // la fonction main exécute la fonction asynchrone [initSession]
47. async function main() {
48.   try {
49.     console.log("requête HTTP vers le serveur en cours ----------------------------------------------");
50.     const response = await initSession(axios);
51.     console.log("succès -----------------------------------------------");
52.     console.log("réponse=", response, typeof (response))
53.   } catch (error) {
54.     console.log("erreur -----------------------------------------------");
55.     console.log("erreur=", error.message);
56.   }
57. }
58.
59. // test
60. main();
```

Commentaires

- ligne 2 : on importe la bibliothèque **[axios]** ;
- lignes 5-6 : configuration par défaut des requêtes HTTP. Les options **[axios.defaults]** sont valables pour toutes les requêtes HTTP émises par l'objet **[axios]** sans qu'on ait besoin de les rappeler à chaque nouvelle requête ;
- ligne 5 : timeout de 2 secondes pour toutes les requêtes ;
- ligne 6 : toutes les URL seront exprimées relativement à l'URL de base ;
- ligne 9 : la fonction asynchrone **[initSession]** ;
- lignes 11-18 : les options de la requête HTTP qui va être émise (en plus des options par défaut déjà définies aux lignes 5-6) ;
- lignes 14-17 : les paramètres de l'URL **[action=init-session&type=json]**. L'objet **[params]** sera automatiquement transformée en chaîne de caractères URL encodée ;
- ligne 22 : appel bloquant de la fonction asynchrone **[axios.request]**. Le 1er paramètre est l'URL cible construite comme **[main.php]** ajouté à l'URL de base définie ligne 6. Le second paramètre est l'objet **[options]** des lignes 11-18 ;
- ligne 25 : **[response]** est un objet Javascript encapsulant la totalité de la réponse HTTP du serveur (entêtes HTTP + document réponse). On l'affiche pour voir sa structure Javascript ;
- ligne 27 : si le serveur a envoyé un document, alors il est trouvé dans **[response.data]**. Ici nous savons que le serveur envoie une réponse jSON accompagnée de l'entête HTTP **[Content-type : application/json]**. La présence de cet entête fait que **[axios]** désérialise automatiquement **[response.data]** en un objet Javascript ;
- ligne 28 : la fonction **[axios]** peut lancer une exception. C'est là que **[axios]** diffère de **[fetch]**. Une exception est lancée dans les cas suivants :
  - la requête HTTP n'a pas pu être émise (erreur côté client) ;
  - le serveur a envoyé un code HTTP d'erreur (400, 404, 500, …) (ce point est en fait configurable). On rappelle que si ce code HTTP est accompagné d'une réponse, **[fetch]** ne lançait pas d'exception alors qu'**[axios]** en lance une. Néanmoins, si le code HTTP d'erreur est accompagné d'un document, celui-ci est mis dans **[error.response]** ;
- ligne 32 : on affiche la structure Javascript de l'objet **[error]** ;
- lignes 33-38 : si l'objet **[error]** contient un objet **[response]** alors c'est cette réponse qu'on rend au code appelant ;
- lignes 39-42 : dans les autres cas, on remonte l'objet **[error]** au code appelant ;
- lignes 47-57 : la fonction asynchrone **[main]** ;
- ligne 50 : appel bloquant à la fonction asynchrone **[initSession]**. On récupère la réponse jSON du serveur comme un objet Javascript ;
- lignes 53-56 : interception de l'erreur éventuelle. Le message d'erreur est dans **[error.message]** ;

Les résultats de l'exécution sont les suivants :

Cas 1 : le serveur Laragon n'est pas lancé

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\http\axios-01.js"
2   requête HTTP vers le serveur en cours ----------------------------------------------
3   axios error= object { Error: timeout of 2000ms exceeded
4       at createError (c:\Data\st-2019\dev\es6\javascript\node_modules\axios\lib\core\createError.js:16:15)
5       at Timeout.handleRequestTimeout (c:\Data\st-
    2019\dev\es6\javascript\node_modules\axios\lib\adapters\http.js:252:16)
6       at ontimeout (timers.js:436:11)
7       at tryOnTimeout (timers.js:300:5)
8       at listOnTimeout (timers.js:263:5)
9       at Timer.processTimers (timers.js:223:10)
```

```
  config:
   { url:
      'http://localhost/php7/scripts-web/impots/version-14/main.php',
     method: 'get',
     params: { action: 'init-session', type: 'json' },
     headers:
      { Accept: 'application/json, text/plain, */*',
        'User-Agent': 'axios/0.19.0' },
     baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
     transformRequest: [ [Function: transformRequest] ],
     transformResponse: [ [Function: transformResponse] ],
     timeout: 2000,
     adapter: [Function: httpAdapter],
     xsrfCookieName: 'XSRF-TOKEN',
     xsrfHeaderName: 'X-XSRF-TOKEN',
     maxContentLength: -1,
     validateStatus: [Function: validateStatus],
     data: undefined },
  code: 'ECONNABORTED',
  request:
   Writable {
     _writableState:
      WritableState {
        objectMode: false,
        highWaterMark: 16384,
        finalCalled: false,
        needDrain: false,
        ending: false,
        ended: false,
        finished: false,
        destroyed: false,
        decodeStrings: true,
        defaultEncoding: 'utf8',
        length: 0,
        writing: false,
        corked: 0,
        sync: true,
        bufferProcessing: false,
        onwrite: [Function: bound onwrite],
        writecb: null,
        writelen: 0,
        bufferedRequest: null,
        lastBufferedRequest: null,
        pendingcb: 0,
        prefinished: false,
        errorEmitted: false,
        emitClose: true,
        bufferedRequestCount: 0,
        corkedRequestsFree: [Object] },
     writable: true,
     domain: null,
     _events:
      [Object: null prototype] {
        response: [Function: handleResponse],
        error: [Function: handleRequestError] },
     _eventsCount: 2,
     _maxListeners: undefined,
     _options:
      { protocol: 'http:',
        maxRedirects: 21,
        maxBodyLength: 10485760,
        path:
         '/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json',
        method: 'GET',
        headers: [Object],
        agent: undefined,
        auth: undefined,
        hostname: 'localhost',
        port: null,
        nativeProtocols: [Object],
        pathname: '/php7/scripts-web/impots/version-14/main.php',
        search: '?action=init-session&type=json' },
     _redirectCount: 0,
     _redirects: [],
     _requestBodyLength: 0,
     _requestBodyBuffers: [],
     _onNativeResponse: [Function],
```

```
87        _currentRequest:
88         ClientRequest {
89           domain: null,
90           _events: [Object],
91           _eventsCount: 6,
92           _maxListeners: undefined,
93           output: [],
94           outputEncodings: [],
95           outputCallbacks: [],
96           outputSize: 0,
97           writable: true,
98           _last: true,
99           chunkedEncoding: false,
100          shouldKeepAlive: false,
101          useChunkedEncodingByDefault: false,
102          sendDate: false,
103          _removedConnection: false,
104          _removedContLen: false,
105          _removedTE: false,
106          _contentLength: 0,
107          _hasBody: true,
108          _trailer: '',
109          finished: true,
110          _headerSent: true,
111          socket: [Socket],
112          connection: [Socket],
113          _header:
114           'GET /php7/scripts-web/impots/version-14/main.php?action=init-
     session&type=json HTTP/1.1\r\nAccept: application/json, text/plain, */*\r\nUser-
     Agent: axios/0.19.0\r\nHost: localhost\r\nConnection: close\r\n\r\n',
115          _onPendingData: [Function: noopPendingOutput],
116          agent: [Agent],
117          socketPath: undefined,
118          timeout: undefined,
119          method: 'GET',
120          path:
121           '/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json',
122          _ended: false,
123          res: null,
124          aborted: 1568528450762,
125          timeoutCb: null,
126          upgradeOrConnect: false,
127          parser: [HTTPParser],
128          maxHeadersCount: null,
129          _redirectable: [Circular],
130          [Symbol(isCorked)]: false,
131          [Symbol(outHeadersKey)]: [Object] },
132       _currentUrl:
133        'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json' },
134    response: undefined,
135    isAxiosError: true,
136    toJSON: [Function] }
137 erreur -----------------------------------------
138 erreur= timeout of 2000ms exceeded
139
140 [Done] exited with code=0 in 2.784 seconds
```

Commentaires

- lignes 33-136 : l'objet **[error]** contient de nombreuses informations ;
  - **[Error]**, lignes 3-9 : une description de l'erreur qui s'est produite ;
  - **[config]**, lignes 10-27 : la configuration de la requête HTTP qui a mené à cette erreur ;
  - **[config.url]**, lignes 11-12 : l'URL cible ;
  - **[config.method]**, ligne 13 : méthode de la requête ;
  - **[config.params]**, ligne 14 : les paramètres de l'URL ;
  - **[config.headers]**, lignes 16-17 : les entêtes HTTP de la requête ;
  - **[config.baseURL]**, ligne 18 : l'URL de base de l'URL cible ;
  - **[config.timeout]**, ligne 21 : le timeout de la requête ;
  - **[code]**, ligne 28 : un code d'erreur ;
  - **[request]**, lignes 29-133 : une description détaillée de la requête HTTP. On remarquera que la plupart des propriétés sont préfixées par l'underscore _ montrant par là que ce sont des propriétés internes à l'objet **[request]** pas destinées à être exploitées directement par le développeur ;
  - **[response]**, ligne 134 : la réponse du serveur, ici inexistante ;
- ligne 138 : le message d'erreur affiché par la fonction **[main]** ;

Cas 2 : le serveur Laragon est lancé

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-2019\dev\es6\javascript\http\axios-01.js"
requête HTTP vers le serveur en cours ----------------------------------------------
réponse axios= { status: 200,
  statusText: 'OK',
  headers:
   { date: 'Sun, 15 Sep 2019 07:09:26 GMT',
     server: 'Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11',
     'x-powered-by': 'PHP/7.2.11',
     'cache-control': 'max-age=0, private, must-revalidate, no-cache, private',
     'set-cookie': [ 'PHPSESSID=uas6lugtblstktcifpd8e5irm6; path=/' ],
     'content-length': '86',
     connection: 'close',
     'content-type': 'application/json' },
  config:
   { url:
      'http://localhost/php7/scripts-web/impots/version-14/main.php',
     method: 'get',
     params: { action: 'init-session', type: 'json' },
     headers:
      { Accept: 'application/json, text/plain, */*',
        'User-Agent': 'axios/0.19.0' },
     baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
     transformRequest: [ [Function: transformRequest] ],
     transformResponse: [ [Function: transformResponse] ],
     timeout: 2000,
     adapter: [Function: httpAdapter],
     xsrfCookieName: 'XSRF-TOKEN',
     xsrfHeaderName: 'X-XSRF-TOKEN',
     maxContentLength: -1,
     validateStatus: [Function: validateStatus],
     data: undefined },
  request:
   ClientRequest {
     domain: null,
     _events:
      [Object: null prototype] {
        socket: [Function],
        abort: [Function],
        aborted: [Function],
        error: [Function],
        timeout: [Function],
        prefinish: [Function: requestOnPrefinish] },
     _eventsCount: 6,
     _maxListeners: undefined,
     output: [],
     outputEncodings: [],
     outputCallbacks: [],
     outputSize: 0,
     writable: true,
     _last: true,
     chunkedEncoding: false,
     shouldKeepAlive: false,
     useChunkedEncodingByDefault: false,
     sendDate: false,
     _removedConnection: false,
     _removedContLen: false,
     _removedTE: false,
     _contentLength: 0,
     _hasBody: true,
     _trailer: '',
     finished: true,
     _headerSent: true,
     socket:
      Socket {
        connecting: false,
        _hadError: false,
        _handle: [TCP],
        _parent: null,
        _host: 'localhost',
        _readableState: [ReadableState],
        readable: true,
        domain: null,
        _events: [Object],
```

```
74          _eventsCount: 7,
75          _maxListeners: undefined,
76          _writableState: [WritableState],
77          writable: false,
78          allowHalfOpen: false,
79          _sockname: null,
80          _pendingData: null,
81          _pendingEncoding: '',
82          server: null,
83          _server: null,
84          parser: null,
85          _httpMessage: [Circular],
86          [Symbol(asyncId)]: 6,
87          [Symbol(lastWriteQueueSize)]: 0,
88          [Symbol(timeout)]: null,
89          [Symbol(kBytesRead)]: 0,
90          [Symbol(kBytesWritten)]: 0 },
91       connection:
92        Socket {
93          connecting: false,
94          _hadError: false,
95          _handle: [TCP],
96          _parent: null,
97          _host: 'localhost',
98          _readableState: [ReadableState],
99          readable: true,
100         domain: null,
101         _events: [Object],
102         _eventsCount: 7,
103         _maxListeners: undefined,
104         _writableState: [WritableState],
105         writable: false,
106         allowHalfOpen: false,
107         _sockname: null,
108         _pendingData: null,
109         _pendingEncoding: '',
110         server: null,
111         _server: null,
112         parser: null,
113         _httpMessage: [Circular],
114         [Symbol(asyncId)]: 6,
115         [Symbol(lastWriteQueueSize)]: 0,
116         [Symbol(timeout)]: null,
117         [Symbol(kBytesRead)]: 0,
118         [Symbol(kBytesWritten)]: 0 },
119      _header:
120       'GET /php7/scripts-web/impots/version-14/main.php?action=init-
    session&type=json HTTP/1.1\r\nAccept: application/json, text/plain, */*\r\nUser-
    Agent: axios/0.19.0\r\nHost: localhost\r\nConnection: close\r\n\r\n',
121      _onPendingData: [Function: noopPendingOutput],
122      agent:
123       Agent {
124         domain: null,
125         _events: [Object],
126         _eventsCount: 1,
127         _maxListeners: undefined,
128         defaultPort: 80,
129         protocol: 'http:',
130         options: [Object],
131         requests: {},
132         sockets: [Object],
133         freeSockets: {},
134         keepAliveMsecs: 1000,
135         keepAlive: false,
136         maxSockets: Infinity,
137         maxFreeSockets: 256 },
138      socketPath: undefined,
139      timeout: undefined,
140      method: 'GET',
141      path:
142       '/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json',
143      _ended: true,
144      res:
145       IncomingMessage {
146         _readableState: [ReadableState],
147         readable: false,
148         domain: null,
```

```
149         _events: [Object],
150         _eventsCount: 3,
151         _maxListeners: undefined,
152         socket: [Socket],
153         connection: [Socket],
154         httpVersionMajor: 1,
155         httpVersionMinor: 0,
156         httpVersion: '1.0',
157         complete: true,
158         headers: [Object],
159         rawHeaders: [Array],
160         trailers: {},
161         rawTrailers: [],
162         aborted: false,
163         upgrade: false,
164         url: '',
165         method: null,
166         statusCode: 200,
167         statusMessage: 'OK',
168         client: [Socket],
169         _consuming: false,
170         _dumped: false,
171         req: [Circular],
172         responseUrl:
173          'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json',
174         redirects: [] },
175      aborted: undefined,
176      timeoutCb: null,
177      upgradeOrConnect: false,
178      parser: null,
179      maxHeadersCount: null,
180      _redirectable:
181       Writable {
182         _writableState: [WritableState],
183         writable: true,
184         domain: null,
185         _events: [Object],
186         _eventsCount: 2,
187         _maxListeners: undefined,
188         _options: [Object],
189         _redirectCount: 0,
190         _redirects: [],
191         _requestBodyLength: 0,
192         _requestBodyBuffers: [],
193         _onNativeResponse: [Function],
194         _currentRequest: [Circular],
195         _currentUrl:
196          'http://localhost/php7/scripts-web/impots/version-14/main.php?action=init-session&type=json' },
197      [Symbol(isCorked)]: false,
198      [Symbol(outHeadersKey)]:
199       [Object: null prototype] { accept: [Array], 'user-agent': [Array], host: [Array] } },
200   data:
201    { action: 'init-session',
202      'état': 700,
203      'réponse': 'session démarrée avec type [json]' } }
204 succès ------------------------------------------
205 réponse= { action: 'init-session',
206   'état': 700,
207   'réponse': 'session démarrée avec type [json]' } object
208
209 [Done] exited with code=0 in 1.115 seconds
```

Commentaires

- lignes 3-203 : l'objet Javascript **[response]** qui encapsule la réponse HTTP du serveur ;
- ligne 3, **[status]** : le code HTTP de la réponse ;
- ligne 4, **[statusText]** : le texte associé au code HTTP précédent ;
- lignes 5-13, **[headers]** : les entêtes HTTP de la réponse :
  - ligne 10, **[Set-Cookie]** : le cookie de session ;
  - ligne 13, **[Content-Type]** : le type du document envoyé par le serveur ;
- lignes 14-31, **[config]** : la configuration de la requête HTTP émise ;
- lignes 32-199, **[request]** : l'objet Javascript détaillant la requête HTTP émise ;
- lignes 200-203, **[request.data]** : l'objet Javascript encapsulant la réponse jSON du serveur ;
- lignes 205-207 : la réponse récupérée par la fonction asynchrone **[main]** ;

Cas 3 : on fait une requête **[init-session]** erronée au serveur Laragon ;

```
8   // init session
9   async function initSession(axios) {
10    // options de la requête HHTP [get /n
11    const options = {
12      method: "GET",
13      // paramètres de l'URL
14      params: {
15        action: 'init-session',
16        type: 'x'
17      }
18    };
```

Les résultats de l'exécution sont les suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\http\axios-01.js"
2   requête HTTP vers le serveur en cours ----------------------------------------------
3   axios error= object { Error: Request failed with status code 400
4      ...
5     config:
6      { url:
7         'http://localhost/php7/scripts-web/impots/version-14/main.php',
8        ...
9        data: undefined },
10    request:
11     ...
12       [Symbol(outHeadersKey)]:
13        [Object: null prototype] { accept: [Array], 'user-agent': [Array], host: [Array] } },
14    response:
15     { status: 400,
16       statusText: 'Bad Request',
17       headers:
18        { date: 'Sun, 15 Sep 2019 07:25:58 GMT',
19          server: 'Apache/2.4.35 (Win64) OpenSSL/1.1.0i PHP/7.2.11',
20          'x-powered-by': 'PHP/7.2.11',
21          'cache-control': 'max-age=0, private, must-revalidate, no-cache, private',
22          'set-cookie': [Array],
23          'content-length': '79',
24          connection: 'close',
25          'content-type': 'application/json' },
26       config:
27        { url:
28           'http://localhost/php7/scripts-web/impots/version-14/main.php',
29          ...
30          data: undefined },
31       request:
32        ...
33          [Symbol(outHeadersKey)]: [Object] },
34       data:
35        { action: 'init-session',
36          'état': 703,
37          'réponse': 'paramètre type=[x] invalide' } },
38    isAxiosError: true,
39    toJSON: [Function] }
40  succès -----------------------------------------------
41  réponse= { action: 'init-session',
42    'état': 703,
43    'réponse': 'paramètre type=[x] invalide' } object
44
45  [Done] exited with code=0 in 0.69 seconds
```

Parce que le serveur a répondu avec un code HTTP 400 (ligne 15), **[axios]** a lancé une exception (encore une fois ce comportement est configurable). Bien que **[axios]** ait lancé une exception, on obtient bien la réponse HTTP du serveur dans **[error.response]** (ligne 14) et le document jSON envoyé **[error.response.data]** (ligne 34). Lignes 41-43 : la fonction **[main]** récupère correctement la réponse jSON du serveur.

2.12.6 script [axios-02]

Maintenant que nous avons détaillé les objets manipulés par la bibliothèque **[axios]** lors d'une requête HTTP, on peut réécrire le script **[axios-01]** de la façon suivante :

```
'use strict';
import axios from 'axios';

// configuration par défaut d'axios
axios.defaults.timeout = 2000;
axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';

// init session
async function initSession(axios) {
  // options de la requête HHTP [get /main.php?action=init-session&type=json]
  const options = {
    method: "GET",
    // paramètres de l'URL
    params: {
      action: 'init-session',
      type: 'json'
    }
  };
  try {
    // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
    const response = await axios.request('main.php', options);
    // la réponse du serveur est dans [response.data]
    return response.data;
  } catch (error) {
    // réponse du serveur
    if (error.response) {
      // la réponse jSON est dans [error.response.data]
      return error.response.data;
    } else {
      // on relance l'erreur
      throw error;
    }
  }
}

// la fonction main exécute la fonction asynchrone [initSession]
async function main() {
  try {
    console.log("requête HTTP vers le serveur en cours ----------------------------------------------");
    const response = await initSession(axios);
    console.log("succès ----------------------------------------------");
    console.log("réponse=", response, typeof (response))
  } catch (error) {
    console.log("erreur ----------------------------------------------");
    console.log("erreur=", error.message);
  }
}

// test
main();
```

2.12.7 script [axios-03]

Le script **[axios-03]** reprend la méthodogie du script **[axios-02]**. On ajoute cette fois la requête HTTP **[authentifier-utilisateur]** vers le serveur qui se fait à l'aide d'un POST :

```
'use strict';
import axios from 'axios';
import qs from 'qs'

// configuration axios
axios.defaults.timeout = 2000;
axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';

// init session
async function initSession(axios) {
  // options de la requête HHTP [get /main.php?action=init-session&type=json]
  const options = {
    method: "GET",
    // paramètres de l'URL
    params: {
      action: 'init-session',
      type: 'json'
    }
  };
```

```
20.   try {
21.     // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
22.     const response = await axios.request('main.php', options);
23.     // la réponse du serveur est dans [response.data]
24.     return response.data;
25.   } catch (error) {
26.     // réponse du serveur
27.     if (error.response) {
28.       // la réponse jSON est dans [error.response.data]
29.       return error.response.data;
30.     } else {
31.       // on relance l'erreur
32.       throw error;
33.     }
34.   }
35. }
36.
37. async function authentifierUtilisateur(axios, user, password) {
38.   // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
39.   const options = {
40.     method: "POST",
41.     headers: {
42.       'Content-type': 'application/x-www-form-urlencoded',
43.     },
44.     // corps du POST
45.     data: qs.stringify({
46.       user: user,
47.       password: password
48.     }),
49.     // paramètres de l'URL
50.     params: {
51.       action: 'authentifier-utilisateur'
52.     }
53.   };
54.   try {
55.     // exécution de la requête HTTP [get /main.php?action=init-session&type=json]
56.     const response = await axios.request('main.php', options);
57.     // la réponse du serveur est dans [response.data]
58.     return response.data;
59.   } catch (error) {
60.     // réponse du serveur
61.     if (error.response) {
62.       // la réponse jSON est dans [error.response.data]
63.       return error.response.data;
64.     } else {
65.       // on relance l'erreur
66.       throw error;
67.     }
68.   }
69. }
70.
71. // la fonction main exécute les fonctions asynchrones une par une
72. async function main() {
73.   try {
74.     // init-session
75.     console.log("action init-session en cours ---------------------------------------------");
76.     const response1 = await initSession(axios);
77.     console.log("succès -------------------------------------------");
78.     console.log("réponse=", response1);
79.     // authentifier-utilisateur
80.     console.log("action authentifier-utilisateur en cours ---------------------------------------------");
81.     const response2 = await authentifierUtilisateur(axios, 'admin', 'admin');
82.     console.log("succès -------------------------------------------");
83.     console.log("réponse=", response2)
84.   } catch (error) {
85.     console.log("erreur -------------------------------------------");
86.     console.log("erreur=", error);
87.   }
88. }
89.
90. // test
91. main();
```

Commentaires

- lignes 38-69 : la fonction asynchrone **[authentifierUtilisateur]** ;

- ligne 38 : il faut faire la requête **[POST /main.php?action=authentifier-utilisateur]** ;
- lignes 39-53 : les options de la requête HTTP ;
- ligne 40 : c'est un POST ;
- lignes 41-43 : les paramètres du POST seront URL encodés dans un document que le client envoie avec sa requête ;
- lignes 45-48 : la propriété **[data]** doit contenir la chaîne du POST URL encodée. Pour cela, on utilise ici la bibliothèque **[qs]** importée ligne 3 ;
- lignes 54-68 : pour l'exécution de la requête, on retrouve le même code que dans la méthode **[initSession]** ;
- lignes 72-88 : la méthode **[asynchrone]** appelle successivement, de manière bloquante, les méthodes **[initSession, authentifierUtilisateur]**, lignes 76 et 81 ;
- ligne 81 : on utilise le couple (admin, admin) comme identifiants de connexion. On sait qu'ils sont reconnus par le serveur ;

Les résultats de l'exécution sont les suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\http\axios-03.js"
2   action init-session en cours ---------------------------------------------
3   succès -------------------------------------------
4   réponse= { action: 'init-session',
5     'état': 700,
6     'réponse': 'session démarrée avec type [json]' }
7   action authentifier-utilisateur en cours ---------------------------------------------
8   succès -------------------------------------------
9   réponse= { action: 'authentifier-utilisateur',
10    'état': 103,
11    'réponse':
12     [ 'pas de session en cours. Commencer par action [init-session]' ] }
13
14  [Done] exited with code=0 in 0.834 seconds
```

- lignes 9-12 : l'authentification utilisateur échoue : le serveur n'a pas retenu le fait qu'on avait initié une session jSON. Cela vient du fait qu'on n'a pas renvoyé le cookie de session envoyé en réponse à la 1ère requête **[init-session]** ;

2.12.8 script [axios-04]

Le script **[axios-04]** amène deux améliorations au script **[axios-03]** :

- il gère le cookie de session ;
- il factorise dans une fonction **[getRemoteData]** ce qui est commun aux fonctions **[initSession]** et **[authentifierUtilisateur]** ;

```
1.  'use strict';
2.  import axios from 'axios';
3.  import qs from 'qs'
4.
5.  // configuration axios
6.  axios.defaults.timeout = 2000;
7.  axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
8.
9.  // cookie de session
10. const sessionCookieName = "PHPSESSID";
11. let sessionCookie = '';
12.
13. // init session
14. async function initSession(axios) {
15.   // options de la requête HHTP [get /main.php?action=init-session&type=json]
16.   const options = {
17.     method: "GET",
18.     // paramètres de l'URL
19.     params: {
20.       action: 'init-session',
21.       type: 'json'
22.     }
23.   };
24.   // exécution de la requête HTTP
25.   return await getRemoteData(axios, options);
26. }
27.
28. async function authentifierUtilisateur(axios, user, password) {
29.   // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
30.   const options = {
31.     method: "POST",
32.     headers: {
```

```
      'Content-type': 'application/x-www-form-urlencoded',
    },
    // corps du POST
    data: qs.stringify({
      user: user,
      password: password
    }),
    // paramètres de l'URL
    params: {
      action: 'authentifier-utilisateur'
    }
  };
  // exécution de la requête HTTP
  return await getRemoteData(axios, options);
}

async function getRemoteData(axios, options) {
  // pour le cookie de session
  if (!options.headers) {
    options.headers = {};
  }
  options.headers.Cookie = sessionCookie;
  // exécution de la requête HTTP
  let response;
  try {
    // requête asynchrone
    response = await axios.request('main.php', options);
  } catch (error) {
    // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
    if (error.response) {
      // la réponse du serveur est dans [error.response]
      response = error.response;
    } else {
      // on relance l'erreur
      throw error;
    }
  }
  // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
  // on récupère le cookie de session s'il existe
  const setCookie = response.headers['set-cookie'];
  if (setCookie) {
    // setCookie est un tableau
    // on cherche le cookie de session dans ce tableau
    let trouvé = false;
    let i = 0;
    while (!trouvé && i < setCookie.length) {
      // on cherche le cookie de session
      const results = RegExp('^(' + sessionCookieName + '.+?);').exec(setCookie[i]);
      if (results) {
        // on mémorise le cookie de session
        // eslint-disable-next-line require-atomic-updates
        sessionCookie = results[1];
        // on a trouvé
        trouvé = true;
      } else {
        // élément suivant
        i++;
      }
    }
  }
  // la réponse du serveur est dans [response.data]
  return response.data;
}

// la fonction main exécute les fonctions asynchrones une par une
async function main() {
  try {
    // init-session
    console.log("action init-session en cours ---------------------------------------------");
    const response1 = await initSession(axios);
    console.log("succès ------------------------------------------");
    console.log("réponse=", response1);
    // authentifier-utilisateur
    console.log("action authentifier-utilisateur en cours ------------------------------------------");
    const response2 = await authentifierUtilisateur(axios, 'admin', 'admin');
    console.log("succès ------------------------------------------");
    console.log("réponse=", response2)
```

```
110.  } catch (error) {
111.    console.log("erreur -------------------------------------------");
112.    console.log("erreur=", error.message);
113.  }
114.}
115.
116.// test
117.main();
```

Commentaires

- lignes 14-26 : la fonction **[initSession]**. Elle se contente désormais de préparer la requête HTTP à envoyer au serveur mais ne l'exécute pas. Elle confie ce rôle à la méthode **[getRemoteDate]** des lignes 49-95 ;
- lignes 28-47 : la fonction **[authentifierUtilisateur]** suit la même démarche ;
- ligne 49 : la fonction **[getRemoteData]** reçoit les deux informations qui lui permettent d'exécuter une requête HTTP:
  - **[axios]**, l'objet qui va se charger d'envoyer la requête et de recevoir la réponse ;
  - **[options]**, les options de configuration de la requête à envoyer au serveur ;
- ligne 59 : exécution de la requête et attente bloquante de sa réponse jSON ;
- lignes 60-68 : gestion de l'éventuelle exception ;
- ligne 64 : on récupère la réponse qui peut être encapsulée dans l'objet d'erreur ;
- ligne 67 : si le serveur a lancé une exception sans y inclure la réponse du serveur, alors on remonte l'erreur reçue au code appelant ;
- la fonction **[getRemoteData]** gère le cookie de session :
  - il le mémorise dans la variable **[sessionCookie]** (ligne 11) lorsqu'il le reçoit la 1ère fois ;
  - il le renvoie ensuite à chaque nouvelle requête HTTP ;
- ligne 72-92 : **[getRemoteData]** analyse chaque réponse du serveur pour savoir s'il a envoyé l'entête HTTP **[Set-Cookie]**. On sait que le serveur envoie un cookie de session nommé **[PHPSESSID]** (ligne 10). C'est donc ce cookie que l'on recherche (ligne 10) ;
- ligne 72 : on récupère les entêtes HTTP **[Set-Cookie]** s'ils existent (la casse n'a pas d'importance). Il peut y avoir en effet plusieurs entêtes **[Set-Cookie]** et c'est donc un tableau que l'on récupère ;
- ligne 73 : si on a récupéré un tableau de cookies ;
- lignes 78-90 : on cherche le cookie de session parmi tous les cookies du tableau ;
- ligne 80 : l'expression relationnelle qui permet de chercher le cookie de session dans le cookie n° i ;
- ligne 81 : si la comparaison a ramené des résultats ;
- ligne 84 : on a dans results**[1]**, la 1ère parenthèse du modèle de l'expression relationnelle, ç-à-d (PHPSESSID=xxxx) jusqu'au ; (non inclus) qui termine le cookie de session ;
- lignes 50-54 : à chaque requête, le cookie de session est inclus dans les entêtes HTTP de la requête. La 1ère fois, ce cookie est vide et sera alors ignoré par le serveur ;

Les résultats de l'exécution sont les suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\http\axios-04.js"
2   action init-session en cours ---------------------------------------------
3   succès -------------------------------------------
4   réponse= { action: 'init-session',
5     'état': 700,
6     'réponse': 'session démarrée avec type [json]' }
7   action authentifier-utilisateur en cours ---------------------------------------------
8   succès -------------------------------------------
9   réponse= { action: 'authentifier-utilisateur',
10    'état': 200,
11    'réponse': 'Authentification réussie [admin, admin]' }
12
13  [Done] exited with code=0 in 0.982 seconds
```

2.13 Les classes

Nous introduisons ici les classes d'ECMAScript 6. Tout d'abord nous montrons que les fonctions peuvent être déjà utilisées comme des classes.

2.13.1 script `[class-00]`

Le script suivant montre une utilisation inhabituelle de fonctions. On les utilise ici comme des objets.

```
1.  'use strict';
2.  // une fonction peut être utilisée comme un objet
3.
4.  // une coquille vide
5.  function f() {
6.
7.  }
8.  // à qui on attribue des propriétés de l'extérieur
9.  f.prop1 = "val1";
10. f.show = function () {
11.   console.log(this.prop1);
12. }
13. // utilisation de f
14. f.show();
15.
16. // une fonction g fonctionnant comme une classe
17. function g() {
18.   this.prop2 = "val2";
19.   this.show = function () {
20.     console.log(this.prop2);
21.   }
22. }
23. // instanciation de la fonction avec [new]
24. new g().show();
```

Commentaires

- lignes 5-7 : le corps de la fonction f ne définit aucune propriété ;
- lignes 9-12 : on donne de l'extérieur des propriétés à la fonction f ;
- ligne 14 : utilisation de la function (objet) f. Notez qu'on n'écrit pas **`[f()]`** mais simplement **`[f]`**. On a là la notation d'un objet ;
- lignes 17-22 : on définit une fonction **`[g]`** comme si c'était une classe avec propriétés et méthodes ;
- ligne 24 : la fonction **`[g]`** est instanciée par **`[new g()]`** ;

Résultats de l'exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\classes\class-00.js"
2   val1
3   val2
```

ES6 a introduit la notion de classe qui nous permet désormais d'éviter de passer par des fonctions pour avoir des classes.

2.13.2 script `[class-01]`

Le script **`[class-01]`** présente une classe **`[Personne]`** :

```
1.  // classe
```

```
2.  class Personne {
3.
4.    // constructeur
5.    constructor(nom, prénom, âge) {
6.      this.nom = nom;
7.      this.prénom = prénom;
8.      this.âge = âge;
9.    }
10.
11.   // getters et setters
12.   get nom() {
13.     return this._nom;
14.   }
15.   set nom(value) {
16.     this._nom = value;
17.   }
18.
19.   get prénom() {
20.     return this._prénom;
21.   }
22.   set prénom(value) {
23.     this._prénom = value;
24.   }
25.
26.   get âge() {
27.     return this._âge;
28.   }
29.   set âge(value) {
30.     this._âge = value;
31.   }
32.
33.   // toString en JSON
34.   toString() {
35.     return JSON.stringify(this);
36.   }
37. }
38.
39. // appel de la classe
40. function main() {
41.   const personne = new Personne("Poirot", "Hercule", 66);
42.   console.log("personne=", personne.toString(), typeof (personne), personne instanceof (Personne));
43. }
44.
45. // appel de main
46. main();
```

Commentaires

- ligne 2 : le mot clé **[class]** désigne une classe ;
- lignes 5-9 : le mot clé **[constructor]** désigne le constructeur de la classe. Il ne peut y en avoir qu'un au plus. Il sert à construire et initialiser une instance de la classe. Notez qu'il n'y a pas de déclaration des propriétés **[nom, prénom, âge]** ;
- lignes 11-36 : propriétés de la classe. On retrouve ici des choses déjà vues dans le paragraphe des objets 4, page 596. Seule la syntaxe diffère ;
- ligne 41 : création d'un objet de type **[Personne]**. A partir de maintenant l'objet **[personne]** s'utilise comme un objet littéral. **[typeof (personne)]** vaut « object » et l'expression **[personne instanceof (Personne)]** est vraie. Il est donc possible de connaître le type exact d'une instance de classe ;

Résultats de l'exécution

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\classes\class-01.js"
2   personne= {"_nom":"Poirot","_prénom":"Hercule","_âge":66} object true
```

2.13.3 script [class-02]

Ce script montre la possibilité d'hériter d'une classe avec le mot clé **[extends]**.

Tout d'abord nous isolons la classe **[Personne]** dans un module **[Personne.js]** :

```
1.  // classe
2.  class Personne {
3.
4.    // constructeur
```

```
5.    constructor(nom, prénom, âge) {
6.      this.nom = nom;
7.      this.prénom = prénom;
8.      this.âge = âge;
9.    }
10.
11.   // getters et setters
12.   get nom() {
13.     return this._nom;
14.   }
15.   set nom(value) {
16.     this._nom = value;
17.   }
18.
19.   get prénom() {
20.     return this._prénom;
21.   }
22.   set prénom(value) {
23.     this._prénom = value;
24.   }
25.
26.   get âge() {
27.     return this._âge;
28.   }
29.   set âge(value) {
30.     this._âge = value;
31.   }
32.
33.   // toString en JSON
34.   toString() {
35.     return JSON.stringify(this);
36.   }
37. }
38. // export classe
39. export default Personne;
```

- ligne 39 : nous exportons la classe **[Personne]** pour que des scripts puissent l'importer ;

Le script **[class-02]** crée une classe **[Enseignant]** dérivée de la classe **[Personne]** :

```
1.  // imports
2.  import Personne from './Personne';
3.
4.  // classe
5.  class Enseignant extends Personne {
6.
7.    // constructeur
8.    constructor(nom, prénom, âge, discipline) {
9.      super(nom, prénom, âge);
10.     this.discipline = discipline;
11.   }
12.
13.   // getters et setters
14.   get discipline() {
15.     return this._discipline;
16.   }
17.   set discipline(value) {
18.     this._discipline = value;
19.   }
20.
21. }
22.
23. // appel de la classe
24. function main() {
25.   const enseignant = new Enseignant("Poirot", "Hercule", 66, "détective");
26.   console.log("enseignant=", enseignant.toString(), typeof (enseignant), enseignant instanceof Enseignant);
27. }
28.
29. // appel de main
30. main();
```

Commentaires

- ligne 2 : on importe la classe **[Personne]** à partir du module **[Personne.js]** qui se trouve dans le même dossier que **[class-02]** ;

- ligne 5 : la classe **[Enseignant]** étend (hérite de) la classe **[Personne]** avec le mot clé **[extends]** : elle lui ajoute une propriété **[_discipline]** avec les getter / setter qui vont avec ;
- lignes 8-11 : le constructeur de la classe **[Enseignant]** reçoit quatre valeurs pour initialiser les quatre propriétés de la classe ;
- ligne 9 : le mot clé **[super]** appelle le constructeur de la classe parent **[Personne]** qui va donc initialiser les propriétés **[_nom, _prénom, _âge]** ;
- ligne 10 : on initialise la propriété **[_discipline]** qui lui appartient à la classe **[Enseignant]** ;
- lignes 14-19 : le getter et le setter de la propriété **[_discipline]** ;
- ligne 25 : on crée un objet de type **[Enseignant]** ;
- ligne 26 : on utilise la méthode **[enseignant.toString()]**. La classe **[Enseignant]** n'a pas cette méthode. C'est alors celle de sa classe parent qui est automatiquement utilisée. Cette méthode rend l'expression **[JSON.stringify(this)]** où **[this]** va être ici un objet **[Enseignant]** et non un objet **[Personne]**. C'est qu'on appelle en programmation objet, le polymorphisme des classes. Un grand mot pour Javascript qui n'est pas un langage orienté objets. Néanmoins, Javascript fait ici ce qu'on attend de lui : il affiche bien un enseignant ;

Les résultats de l'exécution sont les suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\classes\class-02.js"
2   enseignant= {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"} object true
```

- ligne 2 : Javascript reconnaît bien que la variable **[enseignant]** est de type **[Enseignant]** ;

2.13.4 script [class-03]

Le script **[class-03]** montre qu'une classe fille peut redéfinir propriétés et méthodes de sa classe parent. Ici, nous redéfinissons la méthode **[toString]** de la classe parent :

```
1.  // imports
2.  import Personne from './Personne';
3.
4.  // classe
5.  class Enseignant extends Personne {
6.
7.    // constructeur
8.    constructor(nom, prénom, âge, discipline) {
9.      super(nom, prénom, âge);
10.     this.discipline = discipline;
11.   }
12.
13.   // getters et setters
14.   get discipline() {
15.     return this._discipline;
16.   }
17.   set discipline(value) {
18.     this._discipline = value;
19.   }
20.
21.   // redéfinition de toString
22.   toString() {
23.     return "[Enseignant]" + JSON.stringify(this);
24.   }
25. }
26.
27. // appel de la classe
28. function main() {
29.   const enseignant = new Enseignant("Poirot", "Hercule", 66, "détective");
30.   console.log("enseignant=", enseignant.toString(), typeof (enseignant), enseignant instanceof Enseignant);
31. }
32.
33. // appel de main
34. main();
```

Les résultats de l'exécution sont les suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\classes\class-03.js"
2   enseignant= [Enseignant]{"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"} object tr
    ue
```

2.13.5 script [class-04]

Le script **[class-04]** montre de nouveau le polymorphisme à l'oeuvre : là où une fonction attend un paramètre formel de type **[Personne]**, on peut passer un type dérivé tel que **[Enseignant]**. En effet, de par la dérivation, le type **[Enseignant]** a tous les attributs du type **[Personne]**.

Tout d'abord, nous isolons le type **[Enseignant]** dans un module **[Enseignant.js]** :

```
1.  // imports
2.  import Personne from './Personne';
3.
4.  // classe
5.  class Enseignant extends Personne {
6.
7.    // constructeur
8.    constructor(nom, prénom, âge, discipline) {
9.      super(nom, prénom, âge);
10.     this.discipline = discipline;
11.   }
12.
13.   // getters et setters
14.   get discipline() {
15.     return this._discipline;
16.   }
17.   set discipline(value) {
18.     this._discipline = value;
19.   }
20.
21. }
22.
23. // export classe
24. export default Enseignant;
```

- ligne 24 : la classe **[Enseignant]** est exportée pour que d'autres scripts puissent l'importer ;

Le script **[class-04]** est le suivant :

```
1.  // imports
2.  import Enseignant from './Enseignant';
3.  import Personne from './Personne';
4.
5.  // fonction acceptant une personne comme paramètre
6.  function show(personne) {
7.    // dans tous les cas
8.    console.log("paramètre=", personne.toString(), typeof (personne));
9.    // instance de Personne
10.   if (personne instanceof Personne) {
11.     console.log("personne=", personne.toString());
12.   }
13.   // instance de Enseignant
14.   if (personne instanceof Enseignant) {
15.     console.log("enseignant=", personne.toString());
16.   }
17. }
18.
19. // appel de show avec un enseignant
20. show(new Enseignant("Poirot", "Hercule", 66, "détective"));
21. show(new Personne("Marple", "Miss", 70));
```

- ligne 6 : la fonction **[show]** attend un type **[Personne]** ou dérivé ;
- ligne 8 : on affiche la chaîne du paramètre et son type. On va trouver **[object]** ;
- lignes 10-16 : on est capable de savoir si c'est un type **[Personne]** ou un type **[Enseignant]**. Le code peut donc être adapté au type réel du paramètre ;

Les résultats de l'exécution sont les suivants :

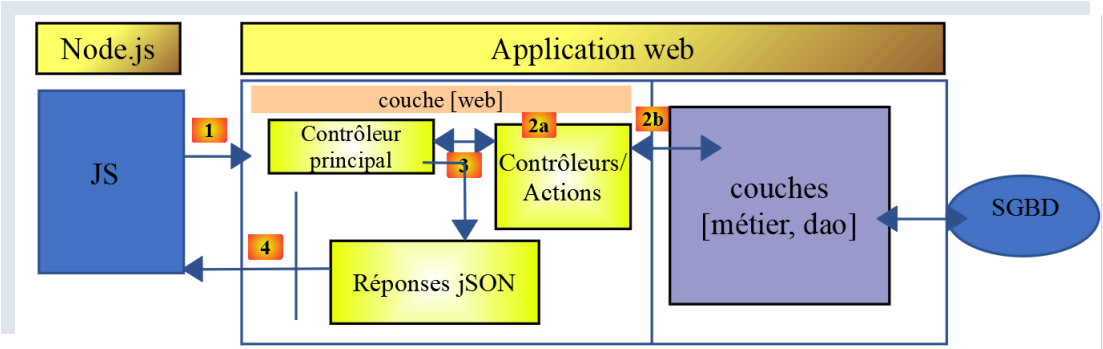
```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\classes\class-04.js"
2   paramètre= {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"} object
3   personne= {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"}
4   enseignant= {"_nom":"Poirot","_prénom":"Hercule","_âge":66,"_discipline":"détective"}
5   paramètre= {"_nom":"Marple","_prénom":"Miss","_âge":70} object
6   personne= {"_nom":"Marple","_prénom":"Miss","_âge":70}
```

- lignes 4 et 6 : Javascript connaît correctement le type des instances de classe ;

2.14 Clients HTTP Javascript du service de calcul de l'impôt

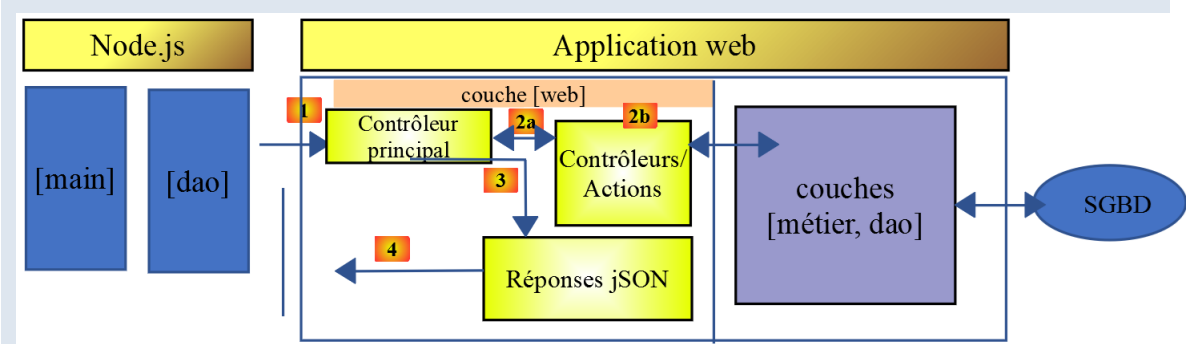
2.14.1 Introduction

Nous nous proposons ici d'écrire un client **[node.js]** de la version 14 du service de calcul de l'impôt. L'architecture client / serveur sera la suivante :

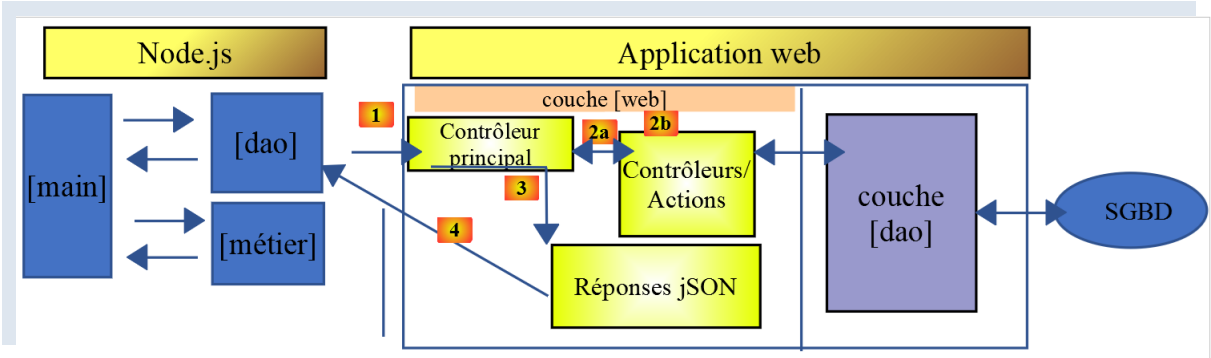


Nous étudierons deux versions du client :

- la version 1 du client aura la structure **[main, dao]** en couches suivante :

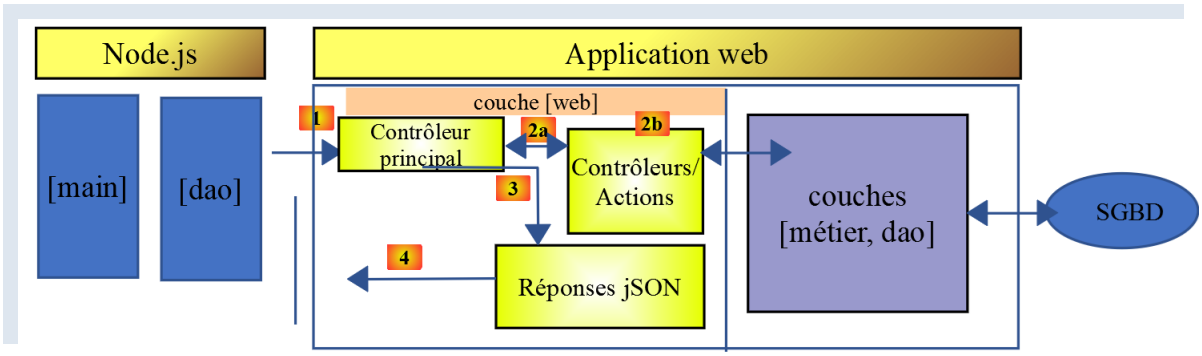


- la version 2 du client aura une structure **[main, métier, dao]**. La couche **[métier]** du serveur sera déportée sur le client :



2.14.2 Client HTTP 1

Comme nous l'avons dit, le client HTTP 1 implémente l'architecture client / serveur suivante :



Nous implémenterons :

- la couche **[dao]** sous la forme d'une classe ;
- la couche **[main]** sous la forme d'un script utilisant cette classe ;

2.14.2.1 La couche [dao]

La couche **[dao]** sera implémentée par la classe suivante **[Dao1.js]** :

```
1.  'use strict';
2.
3.  // imports
4.  import qs from 'qs'
5.
6.  class Dao1 {
7.
8.    // constructeur
9.    constructor(axios) {
10.     // bibliothèque axios pour faire les requêtes HTTP
11.     this.axios = axios;
12.     // cookie de session
13.     this.sessionCookieName = "PHPSESSID";
14.     this.sessionCookie = '';
15.   }
16.
17.   // init session
18.   async  initSession() {
19.     // options de la requête HHTP [get /main.php?action=init-session&type=json]
20.     const options = {
21.       method: "GET",
22.       // paramètres de l'URL
23.       params: {
24.         action: 'init-session',
25.         type: 'json'
26.       }
27.     };
28.     // exécution de la requête HTTP
29.     return await this.getRemoteData(options);
30.   }
31.
32.   async  authentifierUtilisateur(user, password) {
33.     // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
34.     const options = {
35.       method: "POST",
```

```
36.       headers: {
37.         'Content-type': 'application/x-www-form-urlencoded',
38.       },
39.       // corps du POST
40.       data: qs.stringify({
41.         user: user,
42.         password: password
43.       }),
44.       // paramètres de l'URL
45.       params: {
46.         action: 'authentifier-utilisateur'
47.       }
48.     };
49.     // exécution de la requête HTTP
50.     return await this.getRemoteData(options);
51.   }
52.
53.   // calcul de l'impôt
54.   async  calculerImpot(marié, enfants, salaire) {
55.     // options de la requête HHTP
56.     const options = {
57.       method: "POST",
58.       headers: {
59.         'Content-type': 'application/x-www-form-urlencoded',
60.       },
61.       // corps du POST
62.       data: qs.stringify({
63.         marié: marié,
64.         enfants: enfants,
65.         salaire: salaire
66.       }),
67.       // paramètres de l'URL
68.       params: {
69.         action: 'calculer-impot'
70.       }
71.     };
72.     // exécution de la requête HTTP
73.     const data = await this.getRemoteData(options);
74.     // résultat
75.     return data;
76.   }
77.
78.   // liste des simulations
79.   async  listeSimulations() {
80.     // options de la requête HHTP
81.     const options = {
82.       method: "GET",
83.       // paramètres de l'URL
84.       params: {
85.         action: 'lister-simulations'
86.       },
87.     };
88.     // exécution de la requête HTTP
89.     const data = await this.getRemoteData(options);
90.     // résultat
91.     return data;
92.   }
93.
94.   // liste des simulations
95.   async  supprimerSimulation(index) {
96.     // options de la requête HHTP
97.     const options = {
98.       method: "GET",
99.       // paramètres de l'URL
100.      params: {
101.        action: 'supprimer-simulation',
102.        numéro: index
103.      },
104.    };
105.    // exécution de la requête HTTP
106.    const data = await this.getRemoteData(options);
107.    // résultat
108.    return data;
109.  }
110.
111.  async  getRemoteData(options) {
112.    // pour le cookie de session
```

```
113.    if (!options.headers) {
114.      options.headers = {};
115.    }
116.    options.headers.Cookie = this.sessionCookie;
117.    // exécution de la requête HTTP
118.    let response;
119.    try {
120.      // requête asynchrone
121.      response = await this.axios.request('main.php', options);
122.    } catch (error) {
123.      // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
124.      if (error.response) {
125.        // la réponse du serveur est dans [error.response]
126.        response = error.response;
127.      } else {
128.        // on relance l'erreur
129.        throw error;
130.      }
131.    }
132.    // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
133.    // on récupère le cookie de session s'il existe
134.    const setCookie = response.headers['set-cookie'];
135.    if (setCookie) {
136.      // setCookie est un tableau
137.      // on cherche le cookie de session dans ce tableau
138.      let trouvé = false;
139.      let i = 0;
140.      while (!trouvé && i < setCookie.length) {
141.        // on cherche le cookie de session
142.        const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
143.        if (results) {
144.          // on mémorise le cookie de session
145.          // eslint-disable-next-line require-atomic-updates
146.          this.sessionCookie = results[1];
147.          // on a trouvé
148.          trouvé = true;
149.        } else {
150.          // élément suivant
151.          i++;
152.        }
153.      }
154.    }
155.    // la réponse du serveur est dans [response.data]
156.    return response.data;
157.  }
158.}
159.
160.// export de la classe
161.export default Dao1;
```

- nous utilisons ici ce que nous avons appris au paragraphe lien, où nous avons présenté la bibliothèque **[axios]** permettant de faire des requêtes HTTP aussi bien sous **[node.js]** que dans un navigateur. On regardera en particulier le script du paragraphe lien ;
- lignes 9-15 : le constructeur de la classe. Celle-ci aura trois propriétés :
  - **[axios]** : l'objet **[axios]** permettant de faire les requêtes HTTP. Celui-ci est transmis par le code appelant ;
  - **[sessionCookieName]** : selon les serveurs, le cookie de session porte des noms différents. Ici, c'est **[PHPSESSID]** ;
  - **[sessionCookie]** : le cookie de session envoyé par le serveur et mémorisé par le client ;
- lignes 53-76 : la fonction asynchrone **[calculerImpot]** fait la requête **[post /main.php?action=calculer-impot]** en postant les paramètres **[marié, enfants, salaire]**. Elle rend la chaîne jSON transmise par le serveur sous la forme d'un objet Javascript ;
- lignes 79-92 : la fonction asynchrone **[listeSimulations]** fait la requête **[get /main.php?action=lister-simulations. Elle rend la chaîne jSON transmise par le serveur sous la forme d'un objet Javascript ;**
- **lignes 95-109 : la fonction asynchrone [supprimerSimulation]** fait la requête **[get /main.php?action=supprimer-simulation&numéro=index]**. Elle rend la chaîne jSON transmise par le serveur sous la forme d'un objet Javascript ;
- ligne 121 : on utilise la notation **[this.axios]** car ici, l'objet **[axios]** transmis au constructeur a été mémorisé dans la propriété **[this.axios]** ;
- ligne 161 : la classe **[Dao1]** est exportée pour pouvoir être utilisée ;

2.14.2.2 Le script [main1.js]

Le script **[main1.js]** fait une série d'appels au serveur à l'aide de la classe **[Dao1]** :

- initialisation d'une session jSON ;

- authentification avec **[admin, admin]** ;
- demande trois calculs d'impôts ;
- demande la liste des simulations ;
- supprime l'une d'elles ;

Le code est le suivant :

```
1.  // import axios
2.  import axios from 'axios';
3.  // import de la classe Dao
4.  import Dao from './Dao1';
5.
6.  // fonction asynchrone [main]
7.  async function main() {
8.    // configuration axios
9.    axios.defaults.timeout = 2000;
10.   axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
11.   // instanciation couche [dao]
12.   const dao = new Dao(axios);
13.   // utilisation de la couche [dao]
14.   try {
15.     // init session
16.     log("-----------init-session");
17.     let response = await dao.initSession();
18.     log(response);
19.     // authentification
20.     log("-----------authentifier-utilisateur");
21.     response = await dao.authentifierUtilisateur("admin", "admin");
22.     log(response);
23.     // calculs d'impôt
24.     log("-----------calculer-impot x 3");
25.     response = await Promise.all([
26.       dao.calculerImpot("oui", 2, 45000),
27.       dao.calculerImpot("non", 2, 45000),
28.       dao.calculerImpot("non", 1, 30000)
29.     ]);
30.     log(response);
31.     // liste des simulations
32.     log("-----------liste-des-simulations");
33.     response = await dao.listeSimulations();
34.     log(response);
35.     // suppression d'une simulation
36.     log("-----------suppression simulation n° 1");
37.     response = await dao.supprimerSimulation(1);
38.     log(response);
39.   } catch (error) {
40.     // on logue l'erreur
41.     console.log("erreur=", error.message);
42.   }
43. }
44.
45. // log jSON
46. function log(object) {
47.   console.log(JSON.stringify(object, null, 2));
48. }
49.
50. // exécution
51. main();
```

Commentaires

- ligne 2 : on importe la bibliothèque **[axios]** ;
- ligne 4 : on importe la classe **[Dao]** ;
- ligne 7 : la fonction **[main]** qui dialogue avec le serveur est asynchrone ;
- lignes 9-10 : configuration par défaut des requêtes HTTP qui seront faites au serveur :
  - ligne 9 : **[timeout]** de 2 secondes ;
  - ligne 10 : toutes les URL ont pour préfixe, l'URL base de la version 14 du serveur de calcul de l'impôt ;
- ligne 12 : la couche **[Dao]** est construite. On peut désormais l'utiliser ;
- lignes 46-48 : la fonction **[log]** a pour objet d'afficher la chaîne jSON d'un objet Javascript sous une forme embellie : sous forme verticale avec une indentation de deux espaces (3ième paramètre) ;
- lignes 15-18 : initialisation de la session jSON ;
- lignes 19-22 : authentification ;

- lignes 23-30 : trois calculs d'impôt sont demandés en parallèle. Grâce à **[await Promise.all]**, l'exécution est bloquée tant que les trois résultats n'ont pas été tous obtenus ;
- lignes 31-34 : liste des simulations ;
- lignes 35-38 : suppression d'une simulation ;
- lignes 39-42 : gestion de l'éventuelle exception ;

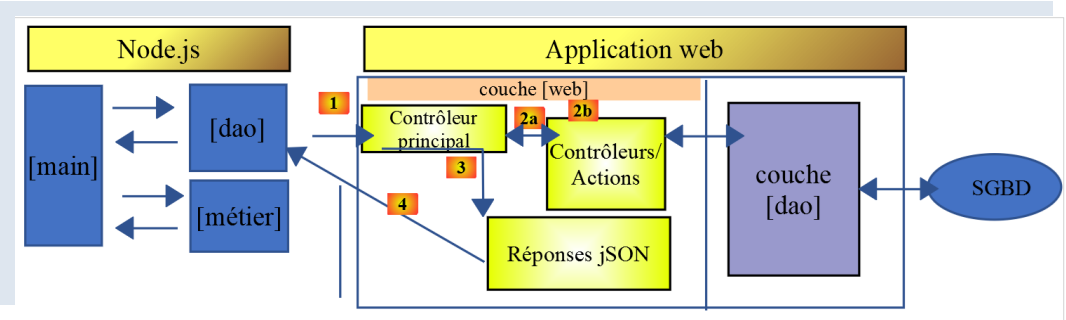
Les résultats de l'exécution sont les suivants :

```
[Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-2019\dev\es6\javascript\client impôts\client http 1\main1.js"
"-----------init-session"
{
  "action": "init-session",
  "état": 700,
  "réponse": "session démarrée avec type [json]"
}
"-----------authentifier-utilisateur"
{
  "action": "authentifier-utilisateur",
  "état": 200,
  "réponse": "Authentification réussie [admin, admin]"
}
"-----------calculer-impot x 3"
[
  {
    "action": "calculer-impot",
    "état": 300,
    "réponse": {
      "marié": "oui",
      "enfants": "2",
      "salaire": "45000",
      "impôt": 502,
      "surcôte": 0,
      "décôte": 857,
      "réduction": 126,
      "taux": 0.14
    }
  },
  {
    "action": "calculer-impot",
    "état": 300,
    "réponse": {
      "marié": "non",
      "enfants": "2",
      "salaire": "45000",
      "impôt": 3250,
      "surcôte": 370,
      "décôte": 0,
      "réduction": 0,
      "taux": 0.3
    }
  },
  {
    "action": "calculer-impot",
    "état": 300,
    "réponse": {
      "marié": "non",
      "enfants": "1",
      "salaire": "30000",
      "impôt": 1687,
      "surcôte": 0,
      "décôte": 0,
      "réduction": 0,
      "taux": 0.14
    }
  }
]
"-----------liste-des-simulations"
{
  "action": "lister-simulations",
  "état": 500,
  "réponse": [
    {
      "marié": "oui",
      "enfants": "2",
      "salaire": "45000",
```

```
68        "impôt": 502,
69        "surcôte": 0,
70        "décôte": 857,
71        "réduction": 126,
72        "taux": 0.14,
73        "arrayOfAttributes": null
74      },
75      {
76        "marié": "non",
77        "enfants": "2",
78        "salaire": "45000",
79        "impôt": 3250,
80        "surcôte": 370,
81        "décôte": 0,
82        "réduction": 0,
83        "taux": 0.3,
84        "arrayOfAttributes": null
85      },
86      {
87        "marié": "non",
88        "enfants": "1",
89        "salaire": "30000",
90        "impôt": 1687,
91        "surcôte": 0,
92        "décôte": 0,
93        "réduction": 0,
94        "taux": 0.14,
95        "arrayOfAttributes": null
96      }
97    ]
98  }
99  "-----------suppression simulation n° 1"
100 {
101   "action": "supprimer-simulation",
102   "état": 600,
103   "réponse": [
104     {
105       "marié": "oui",
106       "enfants": "2",
107       "salaire": "45000",
108       "impôt": 502,
109       "surcôte": 0,
110       "décôte": 857,
111       "réduction": 126,
112       "taux": 0.14,
113       "arrayOfAttributes": null
114     },
115     {
116       "marié": "non",
117       "enfants": "1",
118       "salaire": "30000",
119       "impôt": 1687,
120       "surcôte": 0,
121       "décôte": 0,
122       "réduction": 0,
123       "taux": 0.14,
124       "arrayOfAttributes": null
125     }
126   ]
127 }
128
129 [Done] exited with code=0 in 0.516 seconds
```

2.14.3 Client HTTP 2

L'architecture du client HTTP2 est la suivante :



On a déporté la couche **[métier]** du serveur vers le client Javascript. Contrairement à ce que nous avons pu faire dans le cours PHP7, la couche **[main]** n'aura pas ici à passer par la couche **[métier]** pour atteindre la couche **[dao]**. Nous utiliserons ces deux couches comme des centres de compétences :

- la couche **[main]** passe par la couche **[dao]** dès qu'elle a besoin de données qui sont sur le serveur ;
- la couche **[main]** demande à la couche **[métier]** de faire les calculs de l'impôt ;
- la couche **[métier]** est indépendante de la couche **[dao]** et ne fait jamais appel à elle ;

2.14.3.1 La classe Javascript [Métier]

L'essence de la classe **[Métier]** en PHP a été décrite au paragraphe lien. C'est un code plutôt complexe qu'on rappelle ici, non pour l'expliquer, mais pour pouvoir le traduire en Javascript :

```
1   <?php
2
3   // espace de noms
4   namespace Application;
5
6   class Metier implements InterfaceMetier {
7     // couche Dao
8     private $dao;
9     // données administration fiscale
10    private $taxAdminData;
11
12    //---------------------------------------------
13    // setter couche [dao]
14    public function setDao(InterfaceDao $dao) {
15      $this->dao = $dao;
16      return $this;
17    }
18
19    public function __construct(InterfaceDao $dao) {
20      // on mémorise une référence sur la couche [dao]
21      $this->dao = $dao;
22      // on récupère les données permettant le calcul de l'impôt
23      // la méthode [getTaxAdminData] peut lancer une exception ExceptionImpots
24      // on la laisse alors remonter au code appelant
25      $this->taxAdminData = $this->dao->getTaxAdminData();
26    }
27
28  // calcul de l'impôt
29  // ------------------------------------------------------------------------
30    public function calculerImpot(string $marié, int $enfants, int $salaire): array {
31      // $marié : oui, non
32      // $enfants : nombre d'enfants
33      // $salaire : salaire annuel
34      // $this->taxAdminData : données de l'administration fiscale
35      //
36      // on vérifie qu'on a bien les données de l'administration fiscale
37      if ($this->taxAdminData === NULL) {
38        $this->taxAdminData = $this->getTaxAdminData();
39      }
40      // calcul de l'impôt avec enfants
41      $result1 = $this->calculerImpot2($marié, $enfants, $salaire);
42      $impot1 = $result1["impôt"];
43      // calcul de l'impôt sans les enfants
44      if ($enfants != 0) {
45        $result2 = $this->calculerImpot2($marié, 0, $salaire);
```

```
46        $impot2 = $result2["impôt"];
47        // application du plafonnement du quotient familial
48        $plafonDemiPart = $this->taxAdminData->getPlafondQfDemiPart();
49        if ($enfants < 3) {
50          // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants
51          $impot2 = $impot2 - $enfants * $plafonDemiPart;
52        } else {
53          // $PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants, le double pour les suivants
54          $impot2 = $impot2 - 2 * $plafonDemiPart - ($enfants - 2) * 2 * $plafonDemiPart;
55        }
56      } else {
57        $impot2 = $impot1;
58        $result2 = $result1;
59      }
60      // on prend l'impôt le plus fort
61      if ($impot1 > $impot2) {
62        $impot = $impot1;
63        $taux = $result1["taux"];
64        $surcôte = $result1["surcôte"];
65      } else {
66        $surcôte = $impot2 - $impot1 + $result2["surcôte"];
67        $impot = $impot2;
68        $taux = $result2["taux"];
69      }
70      // calcul d'une éventuelle décôte
71      $décôte = $this->getDecôte($marié, $salaire, $impot);
72      $impot -= $décôte;
73      // calcul d'une éventuelle réduction d'impôts
74      $réduction = $this->getRéduction($marié, $salaire, $enfants, $impot);
75      $impot -= $réduction;
76      // résultat
77      return ["impôt" => floor($impot), "surcôte" => $surcôte, "décôte" => $décôte, "réduction" =>
    $réduction, "taux" => $taux];
78    }
79
80  // ----------------------------------------------------------------------
81    private function calculerImpot2(string $marié, int $enfants, float $salaire): array {
82      // $marié : oui, non
83      // $enfants : nombre d'enfants
84      // $salaire : salaire annuel
85      // $this->taxAdminData : données de l'administration fiscale
86      //
87      // nombre de parts
88      $marié = strtolower($marié);
89      if ($marié === "oui") {
90        $nbParts = $enfants / 2 + 2;
91      } else {
92        $nbParts = $enfants / 2 + 1;
93      }
94      // 1 part par enfant à partir du 3ième
95      if ($enfants >= 3) {
96        // une demi-part de + pour chaque enfant à partir du 3ième
97        $nbParts += 0.5 * ($enfants - 2);
98      }
99      // revenu imposable
100     $revenuImposable = $this->getRevenuImposable($salaire);
101     // surcôte
102     $surcôte = floor($revenuImposable - 0.9 * $salaire);
103     // pour des pbs d'arrondi
104     if ($surcôte < 0) {
105       $surcôte = 0;
106     }
107     // quotient familial
108     $quotient = $revenuImposable / $nbParts;
109     // calcul de l'impôt
110     $limites = $this->taxAdminData->getLimites();
111     $coeffR = $this->taxAdminData->getCoeffR();
112     $coeffN = $this->taxAdminData->getCoeffN();
113     // est mis à la fin du tableau limites pour arrêter la boucle qui suit
114     $limites[count($limites) - 1] = $quotient;
115     // recherche du taux d'imposition
116     $i = 0;
117     while ($quotient > $limites[$i]) {
118       $i++;
119     }
120     // du fait qu'on a placé $quotient à la fin du tableau $limites, la boucle précédente
121     // ne peut déborder du tableau $limites
```

```
122     // maintenant on peut calculer l'impôt
123     $impôt = floor($revenuImposable * $coeffR[$i] - $nbParts * $coeffN[$i]);
124     // résultat
125     return ["impôt" => $impôt, "surcôte" => $surcôte, "taux" => $coeffR[$i]];
126   }
127
128   // revenuImposable=salaireAnnuel-abattement
129   // l'abattement a un min et un max
130   private function getRevenuImposable(float $salaire): float {
131     // abattement de 10% du salaire
132     $abattement = 0.1 * $salaire;
133     // cet abattement ne peut dépasser $this->taxAdminData->getAbattementDixPourCentMax()
134     if ($abattement > $this->taxAdminData->getAbattementDixPourCentMax()) {
135       $abattement = $this->taxAdminData->getAbattementDixPourcentMax();
136     }
137     // l'abattement ne peut être inférieur à $this->taxAdminData->getAbattementDixPourcentMin()
138     if ($abattement < $this->taxAdminData->getAbattementDixPourcentMin()) {
139       $abattement = $this->taxAdminData->getAbattementDixPourcentMin();
140     }
141     // revenu imposable
142     $revenuImposable = $salaire - $abattement;
143     // résultat
144     return floor($revenuImposable);
145   }
146
147 // calcule une décôte éventuelle
148   private function getDecôte(string $marié, float $salaire, float $impots): float {
149     // au départ, une décôte nulle
150     $décôte = 0;
151     // montant maximal d'impôt pour avoir la décôte
152     $plafondImpôtPourDécôte = $marié === "oui" ?
153       $this->taxAdminData->getPlafondImpotCouplePourDecote() :
154       $this->taxAdminData->getPlafondImpotCelibatairePourDecote();
155     if ($impots < $plafondImpôtPourDécôte) {
156       // montant maximal de la décôte
157       $plafondDécôte = $marié === "oui" ?
158         $this->taxAdminData->getPlafondDecoteCouple() :
159         $this->taxAdminData->getPlafondDecoteCelibataire();
160       // décôte théorique
161       $décôte = $plafondDécôte - 0.75 * $impots;
162       // la décôte ne peut dépasser le montant de l'impôt
163       if ($décôte > $impots) {
164         $décôte = $impots;
165       }
166       // pas de décôte <0
167       if ($décôte < 0) {
168         $décôte = 0;
169       }
170     }
171     // résultat
172     return ceil($décôte);
173   }
174
175 // calcule une réduction éventuelle
176   private function getRéduction(string $marié, float $salaire, int $enfants, float $impots): float {
177     // le plafond des revenus pour avoir droit à la réduction de 20%
178     $plafondRevenuPourRéduction = $marié === "oui" ?
179       $this->taxAdminData->getPlafondRevenusCouplePourReduction() :
180       $this->taxAdminData->getPlafondRevenusCelibatairePourReduction();
181     $plafondRevenuPourRéduction += $enfants * $this->taxAdminData->getValeurReducDemiPart();
182     if ($enfants > 2) {
183       $plafondRevenuPourRéduction += ($enfants - 2) * $this->taxAdminData->getValeurReducDemiPart();
184     }
185     // revenu imposable
186     $revenuImposable = $this->getRevenuImposable($salaire);
187     // réduction
188     $réduction = 0;
189     if ($revenuImposable < $plafondRevenuPourRéduction) {
190       // réduction de 20%
191       $réduction = 0.2 * $impots;
192     }
193     // résultat
194     return ceil($réduction);
195   }
196
197   // calcul des impôts en mode batch
```

```
198   public function executeBatchImpots(string $taxPayersFileName, string $resultsFileName, string
    $errorsFileName): void {
199     // on laisse remonter les exceptions qui proviennent de la couche [dao]
200     // on récupère les données contribuables
201     $taxPayersData = $this->dao->getTaxPayersData($taxPayersFileName, $errorsFileName);
202     // tableau des résultats
203     $results = [];
204     // on les exploite
205     foreach ($taxPayersData as $taxPayerData) {
206       // on calcule l'impôt
207       $result = $this->calculerImpot(
208         $taxPayerData->getMarié(),
209         $taxPayerData->getEnfants(),
210         $taxPayerData->getSalaire());
211       // on complète [$taxPayerData]
212       $taxPayerData->setMontant($result["impôt"]);
213       $taxPayerData->setDécôte($result["décôte"]);
214       $taxPayerData->setSurCôte($result["surcôte"]);
215       $taxPayerData->setTaux($result["taux"]);
216       $taxPayerData->setRéduction($result["réduction"]);
217       // on met le résultat dans le tableau des résultats
218       $results [] = $taxPayerData;
219     }
220     // enregistrement des résultats
221     $this->dao->saveResults($resultsFileName, $results);
222   }
223
224 }
```

- lignes 19-26 : le constructeur de la classe PHP. Parce que nous avons dit qu'on construisait une couche **[métier]** indépendante de la couche **[dao]**, nous ferons en Javascript deux modifications à ce constructeur :
  - il ne recevra pas une instance de la couche **[dao]** (il n'en a plus besoin) ;
  - il ne demandera pas les données fiscales de l'administration **[taxAdminData]** à la couche **[dao]** : c'est le code appelant qui transmettra cette donnée au constructeur ;
- lignes 197-122 : nous n'implémenterons pas la méthode **[executeBatchImpots]** dont le but final était d'enregistrer des résultats de simulations dans un fichier texte. Nous voulons un code qui fonctionne à la fois sous **[node.js]** et dans un navigateur. Or sauvegarder des données sur le système de fichiers de la machine exécutant le navigateur client n'est pas possible ;

Avec ces restrictions, le code de la classe Javascript **[Métier]** est le suivant :

```
1   'use strict';
2
3   // classe Métier
4   class Métier {
5
6     // constructeur
7     constructor(taxAdmindata) {
8       // this.taxAdminData : données de l'administration fiscale
9       this.taxAdminData = taxAdmindata;
10    }
11
12    // calcul de l'impôt
13    // ----------------------------------------------------------------------------
14    calculerImpot(marié, enfants, salaire) {
15      // marié : oui, non
16      // enfants : nombre d'enfants
17      // salaire : salaire annuel
18      // this.taxAdminData : données de l'administration fiscale
19      //
20      // calcul de l'impôt avec enfants
21      const result1 = this.calculerImpot2(marié, enfants, salaire);
22      const impot1 = result1["impôt"];
23      // calcul de l'impôt sans les enfants
24      let result2, impot2, plafondDemiPart;
25      if (enfants !== 0) {
26        result2 = this.calculerImpot2(marié, 0, salaire);
27        impot2 = result2["impôt"];
28        // application du plafonnement du quotient familial
29        plafondDemiPart = this.taxAdminData.plafondQfDemiPart;
30        if (enfants < 3) {
31          // PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants
32          impot2 = impot2 - enfants * plafondDemiPart;
33        } else {
34          // PLAFOND_QF_DEMI_PART euros pour les 2 premiers enfants, le double pour les suivants
35          impot2 = impot2 - 2 * plafondDemiPart - (enfants - 2) * 2 * plafondDemiPart;
```

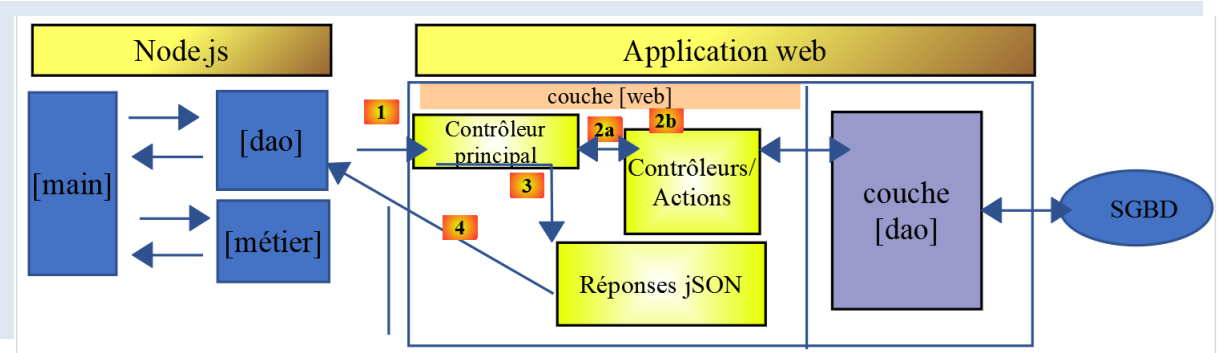
```
36        }
37      } else {
38        // pas de reclacul de l'impôt
39        impot2 = impot1;
40        result2 = result1;
41      }
42      // on prend l'impôt le plus fortdans [impot1, impot2]
43      let impot, taux, surcôte;
44      if (impot1 > impot2) {
45        impot = impot1;
46        taux = result1["taux"];
47        surcôte = result1["surcôte"];
48      } else {
49        surcôte = impot2 - impot1 + result2["surcôte"];
50        impot = impot2;
51        taux = result2["taux"];
52      }
53      // calcul d'une éventuelle décôte
54      const décôte = this.getDecôte(marié, impot);
55      impot -= décôte;
56      // calcul d'une éventuelle réduction d'impôts
57      const réduction = this.getRéduction(marié, salaire, enfants, impot);
58      impot -= réduction;
59      // résultat
60      return {
61        "impôt": Math.floor(impot), "surcôte": surcôte, "décôte": décôte, "réduction": réduction,
62        "taux": taux
63      };
64    }
65
66    // ------------------------------------------------------------------------
67    calculerImpot2(marié, enfants, salaire) {
68      // marié : oui, non
69      // enfants : nombre d'enfants
70      // salaire : salaire annuel
71      // this->taxAdminData : données de l'administration fiscale
72      //
73      // nombre de parts
74      marié = marié.toLowerCase();
75      let nbParts;
76      if (marié === "oui") {
77        nbParts = enfants / 2 + 2;
78      } else {
79        nbParts = enfants / 2 + 1;
80      }
81      // 1 part par enfant à partir du 3ième
82      if (enfants >= 3) {
83        // une demi-part de + pour chaque enfant à partir du 3ième
84        nbParts += 0.5 * (enfants - 2);
85      }
86      // revenu imposable
87      const revenuImposable = this.getRevenuImposable(salaire);
88      // surcôte
89      let surcôte = Math.floor(revenuImposable - 0.9 * salaire);
90      // pour des pbs d'arrondi
91      if (surcôte < 0) {
92        surcôte = 0;
93      }
94      // quotient familial
95      const quotient = revenuImposable / nbParts;
96      // calcul de l'impôt
97      const limites = this.taxAdminData.limites;
98      const coeffR = this.taxAdminData.coeffR;
99      const coeffN = this.taxAdminData.coeffN;
100     // est mis à la fin du tableau limites pour arrêter la boucle qui suit
101     limites[limites.length - 1] = quotient;
102     // recherche du taux d'imposition
103     let i = 0;
104     while (quotient > limites[i]) {
105       i++;
106     }
107     // du fait qu'on a placé quotient à la fin du tableau limites, la boucle précédente
108     // ne peut déborder du tableau limites
109     // maintenant on peut calculer l'impôt
110     const impôt = Math.floor(revenuImposable * coeffR[i] - nbParts * coeffN[i]);
111     // résultat
112     return { "impôt": impôt, "surcôte": surcôte, "taux": coeffR[i] };
```

```
  }

  // revenuImposable=salaireAnnuel-abattement
  // l'abattement a un min et un max
  getRevenuImposable(salaire) {
    // abattement de 10% du salaire
    let abattement = 0.1 * salaire;
    // cet abattement ne peut dépasser taxAdminData.getAbattementDixPourCentMax()
    if (abattement > this.taxAdminData.abattementDixPourCentMax) {
      abattement = this.taxAdminData.abattementDixPourcentMax;
    }
    // l'abattement ne peut être inférieur à taxAdminData.getAbattementDixPourcentMin()
    if (abattement < this.taxAdminData.abattementDixPourcentMin) {
      abattement = this.taxAdminData.abattementDixPourcentMin;
    }
    // revenu imposable
    const revenuImposable = salaire - abattement;
    // résultat
    return Math.floor(revenuImposable);
  }

  // calcule une décôte éventuelle
  getDecôte(marié, impots) {
    // au départ, une décôte nulle
    let décôte = 0;
    // montant maximal d'impôt pour avoir la décôte
    let plafondImpôtPourDécôte = marié === "oui" ?
      this.taxAdminData.plafondImpotCouplePourDecote :
      this.taxAdminData.plafondImpotCelibatairePourDecote;
    let plafondDécôte;
    if (impots < plafondImpôtPourDécôte) {
      // montant maximal de la décôte
      plafondDécôte = marié === "oui" ?
        this.taxAdminData.plafondDecoteCouple :
        this.taxAdminData.plafondDecoteCelibataire;
      // décôte théorique
      décôte = plafondDécôte - 0.75 * impots;
      // la décôte ne peut dépasser le montant de l'impôt
      if (décôte > impots) {
        décôte = impots;
      }
      // pas de décôte <0
      if (décôte < 0) {
        décôte = 0;
      }
    }
    // résultat
    return Math.ceil(décôte);
  }

  // calcule une réduction éventuelle
  getRéduction(marié, salaire, enfants, impots) {
    // le plafond des revenus pour avoir droit à la réduction de 20%
    let plafondRevenuPourRéduction = marié === "oui" ?
      this.taxAdminData.plafondRevenusCouplePourReduction :
      this.taxAdminData.plafondRevenusCelibatairePourReduction;
    plafondRevenuPourRéduction += enfants * this.taxAdminData.valeurReducDemiPart;
    if (enfants > 2) {
      plafondRevenuPourRéduction += (enfants - 2) * this.taxAdminData.valeurReducDemiPart;
    }
    // revenu imposable
    const revenuImposable = this.getRevenuImposable(salaire);
    // réduction
    let réduction = 0;
    if (revenuImposable < plafondRevenuPourRéduction) {
      // réduction de 20%
      réduction = 0.2 * impots;
    }
    // résultat
    return Math.ceil(réduction);
  }
}

// export de la classe
export default Métier;
```

- le code Javascript suit scrupuleusement le code PHP ;

- la classe **[Métier]** est exportée, ligne 187 ;

2.14.3.2 La classe Javascript [Dao2]



La classe **[Dao2]** implémente la couche **[dao]** du client Javascript ci-dessus de la façon suivante :

```
1   'use strict';
2
3   // imports
4   import qs from 'qs'
5
6   class Dao2 {
7
8     // constructeur
9     constructor(axios) {
10      this.axios = axios;
11      // cookie de session
12      this.sessionCookieName = "PHPSESSID";
13      this.sessionCookie = '';
14    }
15
16    // init session
17    async  initSession() {
18      // options de la requête HHTP [get /main.php?action=init-session&type=json]
19      const options = {
20        method: "GET",
21        // paramètres de l'URL
22        params: {
23          action: 'init-session',
24          type: 'json'
25        }
26      };
27      // exécution de la requête HTTP
28      return await this.getRemoteData(options);
29    }
30
31    async  authentifierUtilisateur(user, password) {
32      // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
33      const options = {
34        method: "POST",
35        headers: {
36          'Content-type': 'application/x-www-form-urlencoded',
37        },
38        // corps du POST
39        data: qs.stringify({
40          user: user,
41          password: password
42        }),
43        // paramètres de l'URL
44        params: {
45          action: 'authentifier-utilisateur'
46        }
47      };
48      // exécution de la requête HTTP
49      return await this.getRemoteData(options);
50    }
51
52    async getAdminData() {
53      // options de la requête HHTP [get /main.php?action=get-admindata]
54      const options = {
55        method: "GET",
```

```
56        // paramètres de l'URL
57        params: {
58          action: 'get-admindata'
59        }
60      };
61      // exécution de la requête HTTP
62      const data = await this.getRemoteData(options);
63      // résultat
64      return data;
65    }
66
67    async  getRemoteData(options) {
68      // pour le cookie de session
69      if (!options.headers) {
70        options.headers = {};
71      }
72      options.headers.Cookie = this.sessionCookie;
73      // exécution de la requête HTTP
74      let response;
75      try {
76        // requête asynchrone
77        response = await this.axios.request('main.php', options);
78      } catch (error) {
79        // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
80        if (error.response) {
81          // la réponse du serveur est dans [error.response]
82          response = error.response;
83        } else {
84          // on relance l'erreur
85          throw error;
86        }
87      }
88      // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
89      // on récupère le cookie de session s'il existe
90      const setCookie = response.headers['set-cookie'];
91      if (setCookie) {
92        // setCookie est un tableau
93        // on cherche le cookie de session dans ce tableau
94        let trouvé = false;
95        let i = 0;
96        while (!trouvé && i < setCookie.length) {
97          // on cherche le cookie de session
98          const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
99          if (results) {
100           // on mémorise le cookie de session
101           // eslint-disable-next-line require-atomic-updates
102           this.sessionCookie = results[1];
103           // on a trouvé
104           trouvé = true;
105         } else {
106           // élément suivant
107           i++;
108         }
109       }
110     }
111     // la réponse du serveur est dans [response.data]
112     return response.data;
113   }
114 }
115
116 // export de la classe
117 export default Dao2;
```

Commentaires

- la classe **[Dao2]** n'implémente que trois des requêtes possibles vers le serveur de calcul d'impôt :
  - **[init-session]** (lignes 17-29) : pour initialiser la session jSON ;
  - **[authentifier-utilisateur]** (lignes 31-50) : pour s'authentifier ;
  - **[get-admindata]** (lignes 52-65) : pour avoir les données de l'administration fiscale qui vont permettre de faire les calculs de l'impôt, côté client ;
- lignes 52-65 : nous introduisons une nouvelle action **[get-admindata]** vers le serveur. Cette action n'était pas jusqu'alors implémentée. Nous le faisons maintenant.

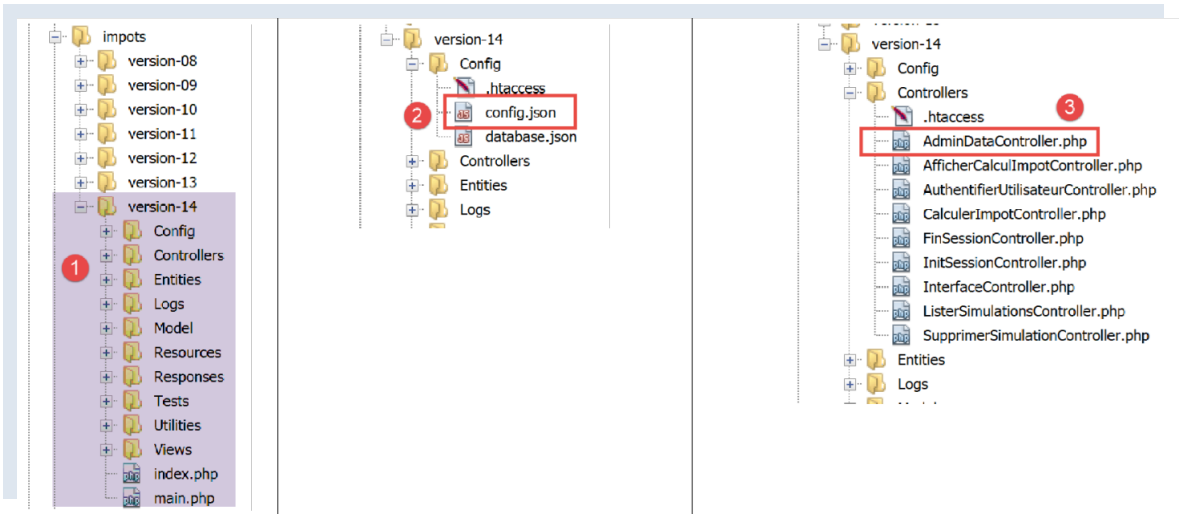
2.14.3.3 Modification du serveur de calcul de l'impôt

Le serveur de calcul de l'impôt doit implémenter une nouvelle action. Nous allons le faire sur la version 14 du serveur. L'action à implémenter a les caractéristiques suivantes :

- elle est demandée par une opération **`[get /main.php?action=get-admindata]`** ;
- elle rend la chaîne jSON d'un objet encapsulant les données de l'administration fiscale ;

Nous allons revoir comment ajouter une action à notre serveur.

La modification se fera sous Netbeans :



En **`[2]`**, nous modifions le fichier **`[config.json]`** pour ajouter la nouvelle action :

```
{
    "databaseFilename": "Config/database.json",
    "corsAllowed": true,
    "rootDirectory": "C:/myprograms/laragon-lite/www/php7/scripts-web/impots/version-14",
    "relativeDependencies": [

        "/Entities/BaseEntity.php",
        "/Entities/Simulation.php",
        "/Entities/Database.php",
        "/Entities/TaxAdminData.php",
        "/Entities/ExceptionImpots.php",

        "/Utilities/Logger.php",
        "/Utilities/SendAdminMail.php",

        "/Model/InterfaceServerDao.php",
        "/Model/ServerDao.php",
        "/Model/ServerDaoWithSession.php",
        "/Model/InterfaceServerMetier.php",
        "/Model/ServerMetier.php",

        "/Responses/InterfaceResponse.php",
        "/Responses/ParentResponse.php",
        "/Responses/JsonResponse.php",
        "/Responses/XmlResponse.php",
        "/Responses/HtmlResponse.php",

        "/Controllers/InterfaceController.php",
        "/Controllers/InitSessionController.php",
        "/Controllers/ListerSimulationsController.php",
        "/Controllers/AuthentifierUtilisateurController.php",
        "/Controllers/CalculerImpotController.php",
        "/Controllers/SupprimerSimulationController.php",
        "/Controllers/FinSessionController.php",
        "/Controllers/AfficherCalculImpotController.php",
        "/Controllers/AdminDataController.php"
    ],
```

```
38      "absoluteDependencies": [
39          "C:/myprograms/laragon-lite/www/vendor/autoload.php",
40          "C:/myprograms/laragon-lite/www/vendor/predis/predis/autoload.php"
41      ],
42      "users": [
43          {
44              "login": "admin",
45              "passwd": "admin"
46          }
47      ],
48      "adminMail": {
49          "smtp-server": "localhost",
50          "smtp-port": "25",
51          "from": "guest@localhost",
52          "to": "guest@localhost",
53          "subject": "plantage du serveur de calcul d'impôts",
54          "tls": "FALSE",
55          "attachments": []
56      },
57      "logsFilename": "Logs/logs.txt",
58      "actions":
59              {
60                  "init-session": "\\InitSessionController",
61                  "authentifier-utilisateur": "\\AuthentifierUtilisateurController",
62                  "calculer-impot": "\\CalculerImpotController",
63                  "lister-simulations": "\\ListerSimulationsController",
64                  "supprimer-simulation": "\\SupprimerSimulationController",
65                  "fin-session": "\\FinSessionController",
66                  "afficher-calcul-impot": "\\AfficherCalculImpotController",
67                  "get-admindata": "\\AdminDataController"
68              },
69      "types": {
70          "json": "\\JsonResponse",
71          "html": "\\HtmlResponse",
72          "xml": "\\XmlResponse"
73      },
74      "vues": {
75          "vue-authentification.php": [700, 221, 400],
76          "vue-calcul-impot.php": [200, 300, 341, 350, 800],
77          "vue-liste-simulations.php": [500, 600]
78      },
79      "vue-erreurs": "vue-erreurs.php"
80  }
```

La modification consiste :

- ligne 67 : ajouter l'action **[get-admindata]** et l'associer à un contrôleur ;
- ligne 36 : déclarer ce contrôleur dans la liste des classes à charger par l'application PHP ;

La phase suivante est d'implémenter le contrôleur **[AdminDataController] [3]** :

```
1   <?php
2
3   namespace Application;
4
5   // dépendances Symfony
6   use \Symfony\Component\HttpFoundation\Response;
7   use \Symfony\Component\HttpFoundation\Request;
8   use \Symfony\Component\HttpFoundation\Session\Session;
9   // alias de la couche [dao]
10  use \Application\ServerDaoWithSession as ServerDaoWithRedis;
11
12  class AdminDataController implements InterfaceController {
13
14    // $config est la configuration de l'application
15    // traitement d'une requête Request
16    // utile la session Session et peut la modifier
17    // $infos sont des informations supplémentaires propres à chaque contrôleur
18    // rend un tableau [$statusCode, $état, $content, $headers]
19    public function execute(
20      array $config,
21      Request $request,
22      Session $session,
23      array $infos = NULL): array {
24
25      // on doit avoir un unique paramètre GET
```

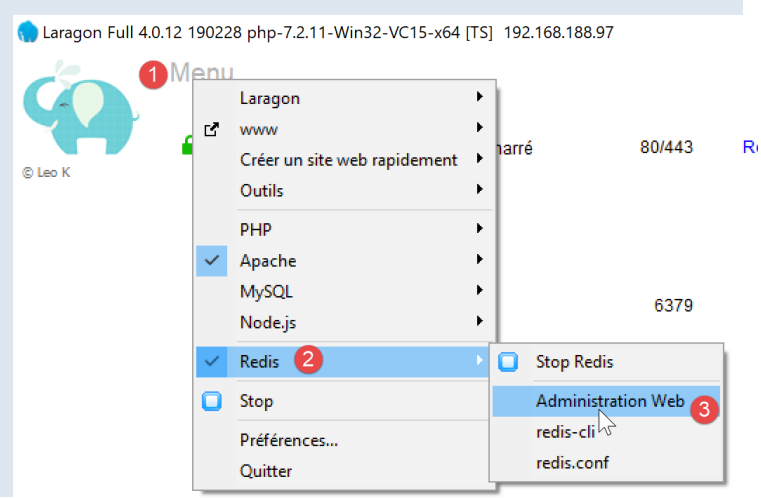
```
26      $method = strtolower($request->getMethod());
27      $erreur = $method !== "get" || $request->query->count() != 1;
28      if ($erreur) {
29        // on note l'erreur
30        $message = "il faut utiliser la méthode [get] avec l'unique paramètre [action] dans l'URL";
31        $état = 1001;
32        // retour résultat au contrôleur principal
33        return [Response::HTTP_BAD_REQUEST, $état, ["réponse" => $message], []];
34      }
35
36      // on peut travailler
37      // Redis
38      \Predis\Autoloader::register();
39      try {
40        // client [predis]
41        $redis = new \Predis\Client();
42        // on se connecte au serveur pour voir s'il est là
43        $redis->connect();
44      } catch (\Predis\Connection\ConnectionException $ex) {
45        // ça s'est mal passé
46        // retour résultat avec erreur au contrôleur principal
47        $état = 1050;
48        return [Response::HTTP_INTERNAL_SERVER_ERROR, $état,
49          ["réponse" => "[redis], " . utf8_encode($ex->getMessage())], []];
50      }
51
52      // récupération des données de l'administration fiscale
53      // on cherche d'abord dans le cache [redis]
54      if (!$redis->get("taxAdminData")) {
55        try {
56          // on va chercher les données fiscales en base de données
57          $dao = new ServerDaoWithRedis($config["databaseFilename"], NULL);
58          // taxAdminData
59          $taxAdminData = $dao->getTaxAdminData();
60          // on met dans redis les données récupérées
61          $redis->set("taxAdminData", $taxAdminData);
62        } catch (\RuntimeException $ex) {
63          // ça s'est mal passé
64          // retour résultat avec erreur au contrôleur principal
65          $état = 1041;
66          return [Response::HTTP_INTERNAL_SERVER_ERROR, $état,
67            ["réponse" => utf8_encode($ex->getMessage())], []];
68        }
69      } else {
70        // les données fiscales sont prises dans la mémoire [redis] de portée [application]
71        $arrayOfAttributes = \json_decode($redis->get("taxAdminData"), true);
72        // on instancie un objet [TaxAdminData] à partir du tableau d'attributs précédent
73        $taxAdminData = (new TaxAdminData())->setFromArrayOfAttributes($arrayOfAttributes);
74      }
75
76      // retour résultat au contrôleur principal
77      $état = 1000;
78      return [Response::HTTP_OK, $état, ["réponse" => $taxAdminData], []];
79    }
80
81  }
```

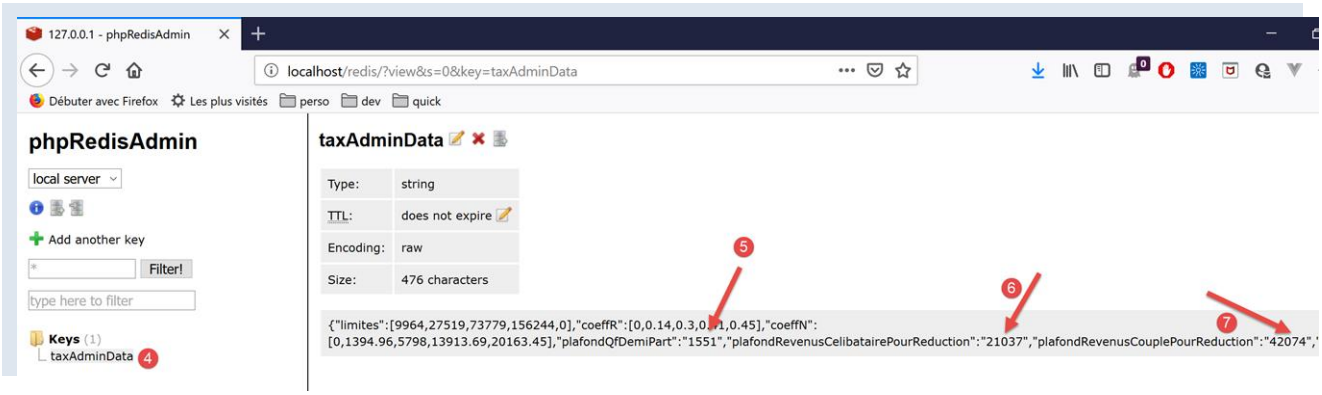
Commentaires

- ligne 12 : comme les autres contrôleurs du serveur, **[AdminDataController]** implémente l'interface **[InterfaceController]** constituée par la méthode **[execute]** des lignes 19-79 ;
- ligne 78 : comme pour les autres contrôleurs du serveur, la méthode **[AdminDataController.execute]** rend un tableau **[$status, $état, ['réponse'=>$response]]** avec :
  - **[$status]** : le code de statut de la réponse HTTP ;
  - **[$état]** : un code interne à l'application représentant l'état dans lequel se trouve le serveur après exécution de la requête du client ;
  - **[$response]** : un tableau encapsulant la réponse à envoyer au client. Ici, ce tableau sera ultérieurement transformé en chaîne jSON ;
- lignes 25-34 : on vérifie que l'action **[get-admindata]** du client est syntaxiquement correcte ;
- lignes 37-74 : on récupère un objet **[TaxAdminData]** trouvé soit :
  - lignes 56-59 : dans la base de données si on ne l'a pas trouvé dans le cache **[redis]** ;
  - lignes 70-73 : dans le cache **[redis]** ;

Ce code reprend celui du contrôleur **[CalculerImpotController]** expliqué dans l'article lien. En effet, ce contrôleur devait lui aussi récupérer l'objet **[TaxAdminData]** encapsulant les données de l'administration fiscale.

Lors des tests du client Javascript, la forme jSON de **[TaxAdminData]** a posé problème lorsque cet objet était trouvé dans le cache **[redis]**. Pour le comprendre, examinons sous quelle forme cet objet est stocké dans **[redis]** :





- en **[5-7]**, on voit que des valeurs numériques ont été stockées sous forme de chaînes de caractères. PHP s'en est accommodé car l'opérateur + dans les calculs entre nombres et chaînes provoque implicitement un changement de type de la chaîne vers un nombre. Mais Javascript fait le contraire : l'opérateur + dans les calculs entre nombres et chaînes provoque implicitement un changement de type du nombre vers une chaîne de caractères. Les calculs de la classe Javascript **[Métier]** sont alors erronés ;

Pour remédier à ce problème, nous modifions la méthode **[TaxAdminData.setFromArrayOfAttributes]** utilisée ligne 71 du contrôleur pour instancier un objet **[TaxAdminData]** (cf. paragraphe) à partir de la chaîne jSON trouvée dans le cache **[redis]** :

```
1   <?php
2
3   namespace Application;
4
5   class TaxAdminData extends BaseEntity {
6     // tranches d'impôt
7     protected $limites;
8     protected $coeffR;
9     protected $coeffN;
10    // constantes de calcul de l'impôt
11    protected $plafondQfDemiPart;
12    protected $plafondRevenusCelibatairePourReduction;
13    protected $plafondRevenusCouplePourReduction;
14    protected $valeurReducDemiPart;
15    protected $plafondDecoteCelibataire;
16    protected $plafondDecoteCouple;
17    protected $plafondImpotCouplePourDecote;
18    protected $plafondImpotCelibatairePourDecote;
```

```php
  protected $abattementDixPourcentMax;
  protected $abattementDixPourcentMin;

  // initialisation
  public function setFromJsonFile(string $taxAdminDataFilename) {
    // parent
    parent::setFromJsonFile($taxAdminDataFilename);
    // on vérifie les valeurs des attributs
    $this->checkAttributes();
    // on rend l'objet
    return $this;
  }

  protected function check($value): \stdClass {
    // $value est un tableau d'éléments de type string ou un unique élément
    if (!\is_array($value)) {
      $tableau = [$value];
    } else {
      $tableau = $value;
    }
    // on transforme le tableau de strings en tableau de réels
    $newTableau = [];
    $result = new \stdClass();
    // les éléments du tableau doivent être des nombres décimaux positifs ou nuls
    $modèle = '/^\s*([+]?)\s*(\d+\.\d*|\.\d+|\d+)\s*$/';
    for ($i = 0; $i < count($tableau); $i ++) {
      if (preg_match($modèle, $tableau[$i])) {
        // on met le float dans newTableau
        $newTableau[] = (float) $tableau[$i];
      } else {
        // on note l'erreur
        $result->erreur = TRUE;
        // on quitte
        return $result;
      }
    }
    // on rend le résultat
    $result->erreur = FALSE;
    if (!\is_array($value)) {
      // une seule valeur
      $result->value = $newTableau[0];
    } else {
      // une liste de valeurs
      $result->value = $newTableau;
    }
    return $result;
  }

  // initialisation par un tableau d'attributs
  public function setFromArrayOfAttributes(array $arrayOfAttributes) {
    // parent
    parent::setFromArrayOfAttributes($arrayOfAttributes);
    // on vérifie les valeurs des attributs
    $this->checkAttributes();
    // on rend l'objet
    return $this;
  }

  // vérification des valeurs des attributs
  protected function checkAttributes() {
    // on vérifie que les valeurs des attributs sont des réels >=0
    foreach ($this as $key => $value) {
      if (is_string($value)) {
        // $value doit être un nbre réel >=0 ou un tableau de réels >=0
        $result = $this->check($value);
        // erreur ?
        if ($result->erreur) {
          // on lance une exception
          throw new ExceptionImpots("La valeur de l'attribut [$key] est invalide");
        } else {
          // on note la valeur
          $this->$key = $result->value;
        }
      }
    }

    // on rend l'objet
```

```
96      return $this;
97    }
98
99    // getters et setters
100   ...
101
102 }
```

Commentaires

- ligne 5 : la classe **[TaxAdminData]** étend la classe **[BaseEntity]** qui a déjà la méthode **[setFromArrayOfAttributes]**. Celle-ci ne convenant pas, nous la redéfinissons aux lignes 67-75 ;
- ligne 70 : la méthode **[setFromArrayOfAttributes]** de la classe parent est d'abord utilisée pour initialiser les attributs de la classe ;
- ligne 72 : la méthode **[checkAttributes]** vérifie que les valeurs associées sont bien des nombres. Si ce sont des chaînes, celles-ci sont converties en nombres ;
- ligne 74 : l'objet **[$this]** rendu est alors un objet avec des attributs à valeurs numériques ;
- lignes 78-93 : la méthode **[checkAttributes]** vérifie que les valeurs associées aux attributs de l'objet sont bien numériques ;
- ligne 80 : on parcourt la liste des attributs ;
- ligne 81 : si la valeur d'un attribut est de type **[string]** ;
- ligne 83 : alors on vérifie que cette chaîne représente un nombre ;
- ligne 90 : si c'est le cas, la chaîne est transformée en nombre et affectée à l'attribut testé ;
- lignes 85-86 : si ce n'est pas le cas, une exception est lancée ;
- lignes 32-65 : la fonction **[check]** fait un peu plus que nécessaire. Elle traite aussi bien des tableaux que des valeurs uniques. Or ici, elle n'est appelée que pour vérifier une valeur de type **[string]**. Elle rend un objet avec les propriétés **[erreur, value]** où :
  - **[erreur]** est un booléen signalant une erreur ou non ;
  - **[value]** est le paramètre **[value]** de la ligne 32, transformée en nombre ou tableau de nombres selon les cas ;

La classe **[BaseEntity]** qui pouvait avoir un attribut nommé **[arrayOfAttributes]** est modifiée pour ne plus avoir celui-ci : il pollue en effet la chaîne jSON de **[TaxAdminData]**. La classe est réécrite de la façon suivante :

```
1   <?php
2
3   namespace Application;
4
5   class BaseEntity {
6
7     // initialisation à partir d'un fichier JSON
8     public function setFromJsonFile(string $jsonFilename) {
9       // on récupère le contenu du fichier des données fiscales
10      $fileContents = \file_get_contents($jsonFilename);
11      $erreur = FALSE;
12      // erreur ?
13      if (!$fileContents) {
14        // on note l'erreur
15        $erreur = TRUE;
16        $message = "Le fichier des données [$jsonFilename] n'existe pas";
17      }
18      if (!$erreur) {
19        // on récupère le code JSON du fichier de configuration dans un tableau associatif
20        $arrayOfAttributes = \json_decode($fileContents, true);
21        // erreur ?
22        if ($arrayOfAttributes === FALSE) {
23          // on note l'erreur
24          $erreur = TRUE;
25          $message = "Le fichier de données JSON [$jsonFilename] n'a pu être exploité correctement";
26        }
27      }
28      // erreur ?
29      if ($erreur) {
30        // on lance une exception
31        throw new ExceptionImpots($message);
32      }
33      // initialisation des attributs de la classe
34      foreach ($arrayOfAttributes as $key => $value) {
35        $this->$key = $value;
36      }
37      // on vérifie la présence de tous les attributs
38      $this->checkForAllAttributes($arrayOfAttributes);
39      // on rend l'objet
40      return $this;
```

```
41    }
42
43    public function checkForAllAttributes($arrayOfAttributes) {
44      // on vérifie que toutes les clés ont été initialisées
45      foreach (\array_keys($arrayOfAttributes) as $key) {
46        if (!isset($this->$key)) {
47          throw new ExceptionImpots("L'attribut [$key] de la classe "
48            . get_class($this) . " n'a pas été initialisé");
49        }
50      }
51    }
52
53    public function setFromArrayOfAttributes(array $arrayOfAttributes) {
54      // on initialise certains attributs de la classe (pas forcément tous)
55      foreach ($arrayOfAttributes as $key => $value) {
56        $this->$key = $value;
57      }
58      // on retourne l'objet
59      return $this;
60    }
61
62    // toString
63    public function __toString() {
64      // attributs de l'objet
65      $arrayOfAttributes = \get_object_vars($this);
66      // chaîne jSON de l'objet
67      return \json_encode($arrayOfAttributes, JSON_UNESCAPED_UNICODE);
68    }
69
70  }
```

Commentaires

- ligne 20 : l'attribut **[$this→arrayOfAttributes]** a été transformée en variable qui doit être désormais passée à la méthode **[checkForAllAttributes]**, ligne 38 qui auparavant opérait sur l'attribut **[$this→arrayOfAttributes]** ;

A cause de ce changement sur **[BaseEntity]**, la classe **[Database]** doit être également légèrement modifiée :

```
1   <?php
2
3   namespace Application;
4
5   class Database extends BaseEntity {
6     // attributs
7     protected $dsn;
8     protected $id;
9     protected $pwd;
10    protected $tableTranches;
11    protected $colLimites;
12    protected $colCoeffR;
13    protected $colCoeffN;
14    protected $tableConstantes;
15    protected $colPlafondQfDemiPart;
16    protected $colPlafondRevenusCelibatairePourReduction;
17    protected $colPlafondRevenusCouplePourReduction;
18    protected $colValeurReducDemiPart;
19    protected $colPlafondDecoteCelibataire;
20    protected $colPlafondDecoteCouple;
21    protected $colPlafondImpotCelibatairePourDecote;
22    protected $colPlafondImpotCouplePourDecote;
23    protected $colAbattementDixPourcentMax;
24    protected $colAbattementDixPourcentMin;
25
26    // setter
27    // initialisation
28    public function setFromJsonFile(string $jsonFilename) {
29      // parent
30      parent::setFromJsonFile($jsonFilename);
31      // on retourne l'objet
32      return $this;
33    }
34
35    // getters et setters
36    ...
37  }
```

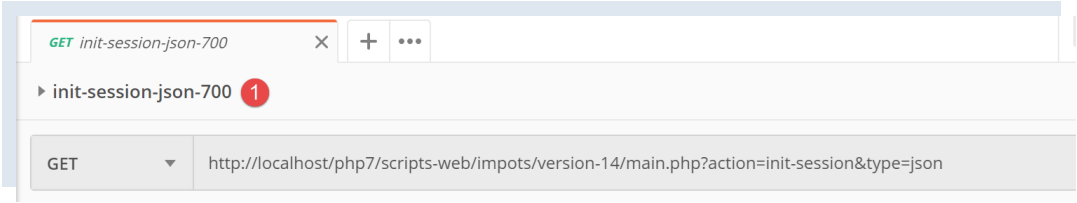
Commentaires

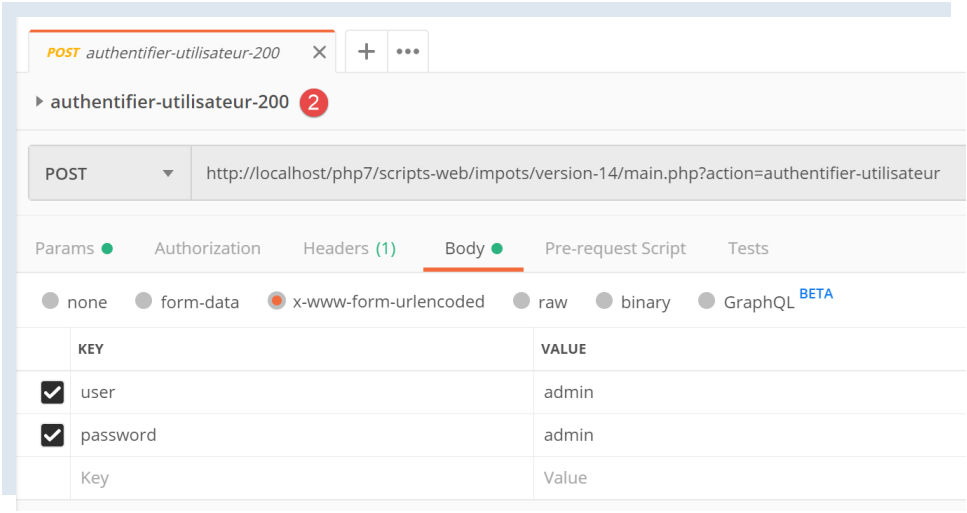
- dans le code original, après la ligne 30, on appelait la méthode **`[parent::checkForAllAttributes]`**. Cela n'a plus à être fait puisque c'est désormais pris automatiquement en charge par la méthode **`[parent::setFromJsonFile($jsonFilename)]`** ;

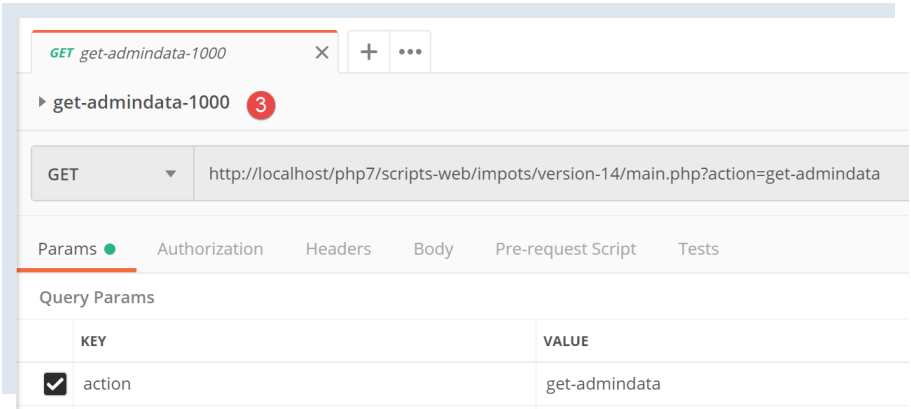
2.14.3.4 Tests `[Postman]` du serveur

`[Postman]` a été présenté dans l'article lien.

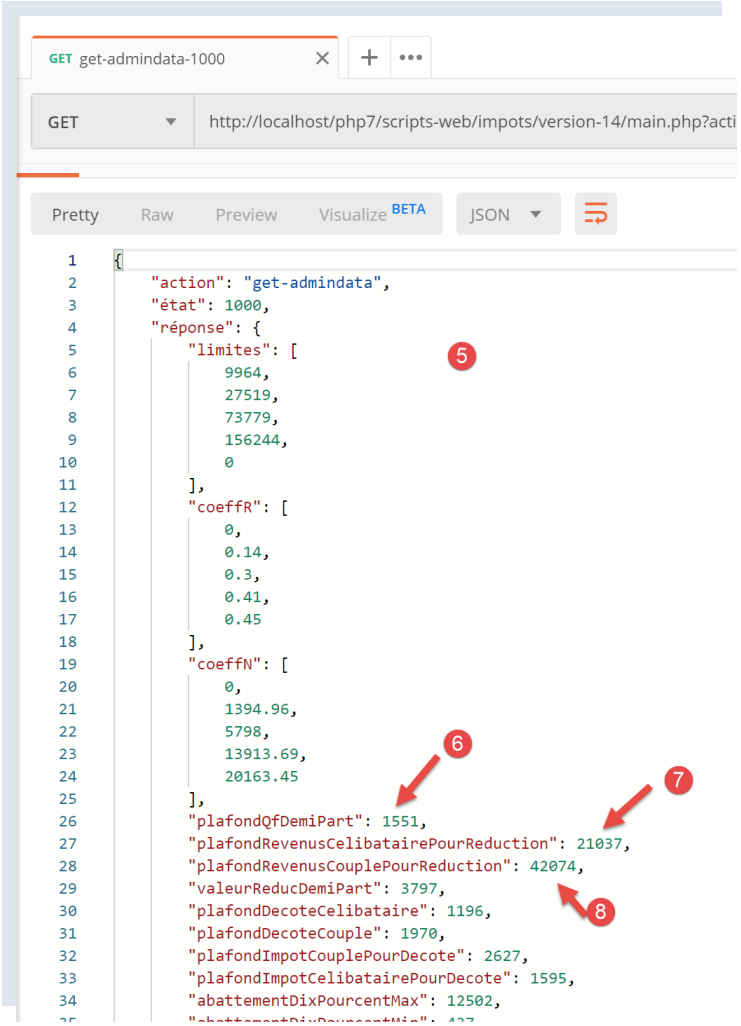
Nous utilisons les tests Postman suivants :





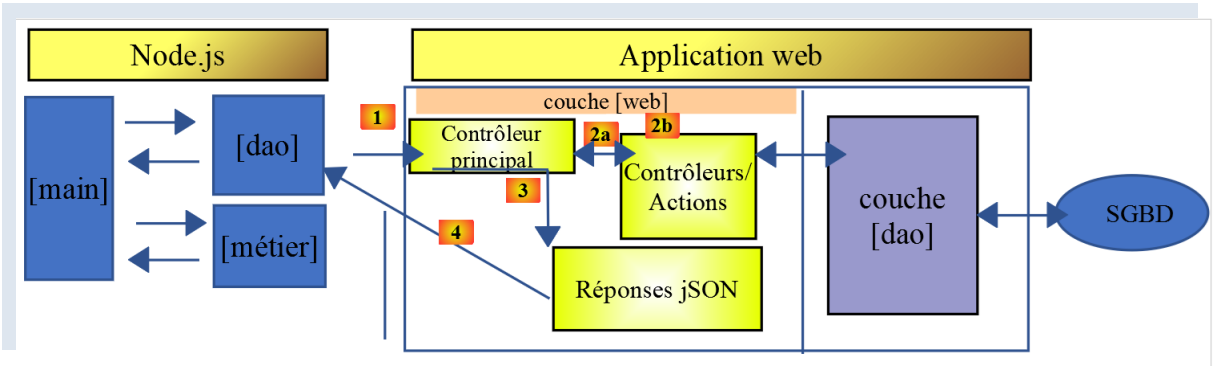


Le résultat jSON de cette dernière requête est la suivante :



- en **[5-8]**, on peut remarquer que les attributs de la chaîne jSON ont bien des valeurs numériques (et non chaînes de caractères). Ce résultat va permettre à la classe Javascript **[Métier]** de s'exécuter normalement ;

2.14.3.5 Le script principal [main]



Le script principal **[main]** du client Javascript est le suivant :

```
1   // imports
2   import axios from 'axios';
3
4   // imports
5   import Dao from './Dao2';
6   import Métier from './Métier';
7
8   // fonction asynchrone [main]
```

```
9   async function main() {
10    // configuration axios
11    axios.defaults.timeout = 2000;
12    axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
13    // instanciation couche [dao]
14    const dao = new Dao(axios);
15    // requêtes HTTP
16    let taxAdminData;
17    try {
18      // init session
19      log("-----------init-session");
20      let response = await dao.initSession();
21      log(response);
22      // authentification
23      log("-----------authentifier-utilisateur");
24      response = await dao.authentifierUtilisateur("admin", "admin");
25      log(response);
26      // données fiscales
27      log("-----------get-admindata");
28      response = await dao.getAdminData();
29      log(response);
30      taxAdminData = response.réponse;
31    } catch (error) {
32      // on logue l'erreur
33      console.log("erreur=", error.message);
34      // fin
35      return;
36    }
37
38    // instanciation couche [métier]
39    const métier = new Métier(taxAdminData);
40
41    // calculs d'impôt
42    log("-----------calculer-impot x 3");
43    const simulations = [];
44    simulations.push(métier.calculerImpot("oui", 2, 45000));
45    simulations.push(métier.calculerImpot("non", 2, 45000));
46    simulations.push(métier.calculerImpot("non", 1, 30000));
47    // liste des simulations
48    log("-----------liste-des-simulations");
49    log(simulations);
50    // suppression d'une simulation
51    log("-----------suppression simulation n° 1");
52    simulations.splice(1, 1);
53    log(simulations);
54  }
55
56  // log jSON
57  function log(object) {
58    console.log(JSON.stringify(object, null, 2));
59  }
60
61  // exécution
62  main();
```

Commentaires

- lignes 5-6 : imports des classes **[Dao]** et **[Métier]** ;
- ligne 9 : la fonction asynchrone **[main]** qui va organiser le dialogue avec le serveur grâce à la classe **[Dao]** et demander à la classe **[Métier]** de faire les calculs d'impôt ;
- lignes 10-36 : le script appelle successivement et de façon bloquante, les méthodes **[initSession, authentifierUtilisateur, getAdminData]** de la couche **[dao]** ;
- ligne 38 : on n'a plus besoin de la couche **[dao]**. On a tous les éléments pour faire travailler la couche **[métier]** du client Javascript ;
- lignes 41-46 : on fait trois calculs d'impôt dont on cumule les résultats dans un tableau **[simulations]** ;
- ligne 49 : on affiche le tableau des simulations ;
- ligne 52 : on supprime l'une d'elles ;

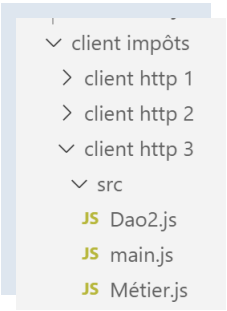
Les résultats de l'exécution du script principal sont les suivants :

```
1   [Running] C:\myprograms\laragon-lite\bin\nodejs\node-v10\node.exe -r esm "c:\Data\st-
    2019\dev\es6\javascript\client impôts\client http 2\main2.js"
2   "-----------init-session"
3   {
```

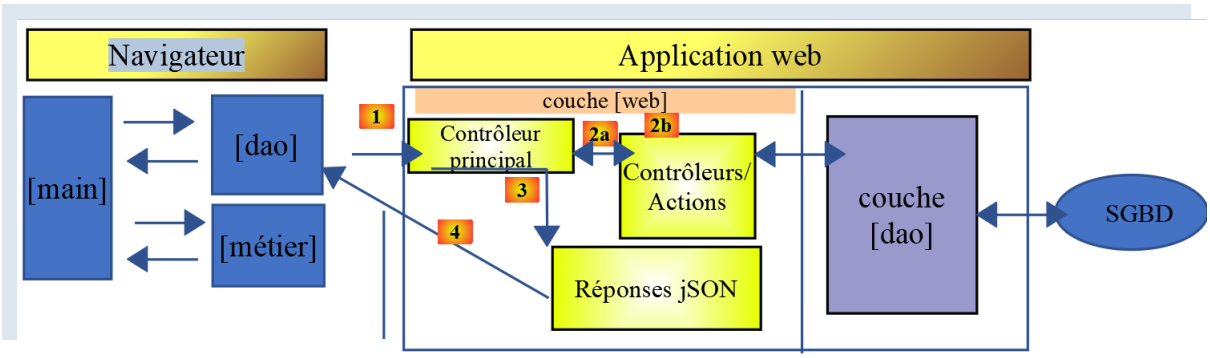
```
    "action": "init-session",
    "état": 700,
    "réponse": "session démarrée avec type [json]"
  }
  "-----------authentifier-utilisateur"
  {
    "action": "authentifier-utilisateur",
    "état": 200,
    "réponse": "Authentification réussie [admin, admin]"
  }
  "-----------get-admindata"
  {
    "action": "get-admindata",
    "état": 1000,
    "réponse": {
      "limites": [
        9964,
        27519,
        73779,
        156244,
        0
      ],
      "coeffR": [
        0,
        0.14,
        0.3,
        0.41,
        0.45
      ],
      "coeffN": [
        0,
        1394.96,
        5798,
        13913.69,
        20163.45
      ],
      "plafondQfDemiPart": 1551,
      "plafondRevenusCelibatairePourReduction": 21037,
      "plafondRevenusCouplePourReduction": 42074,
      "valeurReducDemiPart": 3797,
      "plafondDecoteCelibataire": 1196,
      "plafondDecoteCouple": 1970,
      "plafondImpotCouplePourDecote": 2627,
      "plafondImpotCelibatairePourDecote": 1595,
      "abattementDixPourcentMax": 12502,
      "abattementDixPourcentMin": 437
    }
  }
  "-----------calculer-impot x 3"
  "-----------liste-des-simulations"
  [
    {
      "impôt": 502,
      "surcôte": 0,
      "décôte": 857,
      "réduction": 126,
      "taux": 0.14
    },
    {
      "impôt": 3250,
      "surcôte": 370,
      "décôte": 0,
      "réduction": 0,
      "taux": 0.3
    },
    {
      "impôt": 1687,
      "surcôte": 0,
      "décôte": 0,
      "réduction": 0,
      "taux": 0.14
    }
  ]
  "-----------suppression simulation n° 1"
  [
    {
      "impôt": 502,
```

```
81      "surcôte": 0,
82      "décôte": 857,
83      "réduction": 126,
84      "taux": 0.14
85    },
86    {
87      "impôt": 1687,
88      "surcôte": 0,
89      "décôte": 0,
90      "réduction": 0,
91      "taux": 0.14
92    }
93  ]
94
95  [Done] exited with code=0 in 0.583 seconds
```

2.14.4 Client HTTP 3



Dans cette section, nous portons l'application **[Client HTTP 2]** dans un navigateur selon l'architecture suivane :



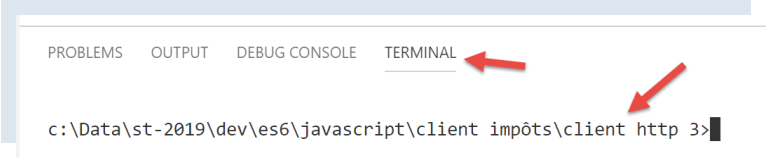
Le portage n'est pas immédiat. Si **[node.js]** sait exécuter du Javascript ES6, ce n'est pas le cas en général des navigateurs. Il faut alors utiliser des outils qui traduisent le code ES6 en code ES5 compris par les navigateurs récents. Heureusement ces outils sont à la fois puissants et plutôt simples d'utilisation.

Nous avons ici suivi l'article |**How to write ES6 code that's safe to run in the browser - Web Developer's Journal**|.

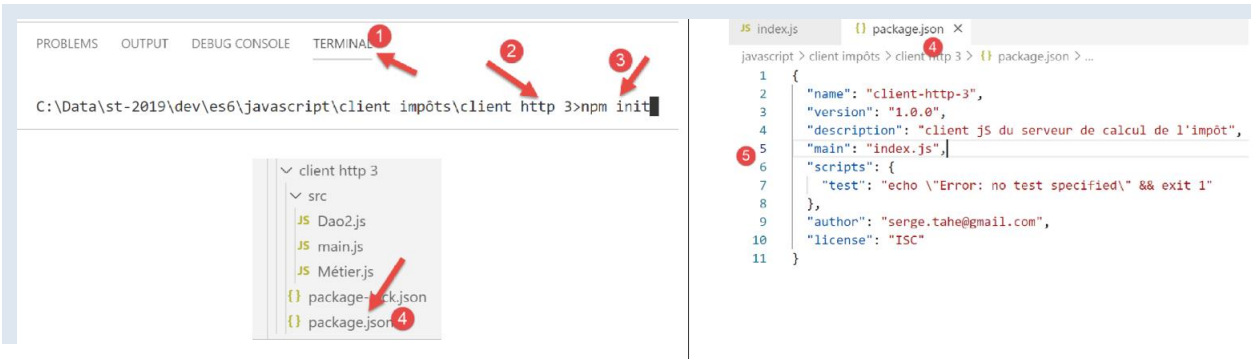
Dans le dossier **[client HTTP 3/src]**, on a mis les éléments **[main.js, Métier.js, Dao2.js]** de l'application **[Client Http 2]** que nous venons de développer.

2.14.4.1 Initialisation du projet

Nous allons travailler dans le dossier **[client http 3]**. Nous ouvrons un terminal dans **[VSCode]** et nous nous positionnons sur ce dossier :



Nous initialisons ce projet avec la commande **[npm init]** et nous acceptons pour les questions posées les réponses proposées par défaut :



- en **[4-5]**, le fichier de configuration du projet **[package.json]** généré à partir des différentes réponses données ;

2.14.4.2 Installation des dépendances du projet

Nous allons installer les dépendances suivantes :

- **[@babel/core]** : le coeur de l'outil **[Babel]** **[https://babeljs.io]** qui transforme du code ES 2015+ en code exécutable sur les navigateurs récents et plus anciens ;
- **[@babel/preset-env]** : fait partie de l'outillage Babel. Intervient avant la transpilation ES6 → ES5 ;
- **[babel-loader]** : cette dépendance permet à l'outil **[webpack]** de faire appel à l'outil **[Babel]** ;
- **[webpack]** : chef d'orchestre. C'est **[webpack]** qui fait appel à Babel pour faire la transpilation des codes ES6 → ES5 puis lui qui assemble la totalité des fichiers résultants dans un unique fichier ;
- **[webpack-cli]** : nécessaire à **[webpack]** ;
- **[@webpack-cli/init]** : utilisé pour configurer **[webpack]** ;
- **[webpack-dev-server]** : fournit un serveur web de développement opérant par défaut sur le port 8080. Lorsque les fichiers sources sont modifiés, recharge automatiquement l'application web ;

Les dépendances du projet sont installées de la façon suivante dans un terminal de **[VSCode]** :

```
npm --save-dev install @babel/core @babel/preset-env babel-loader webpack webpack-cli webpack-dev-server @webpack-cli/init
```

Après l'installation des dépendances, le fichier **[package.json]** a évolué de la façon suivante :

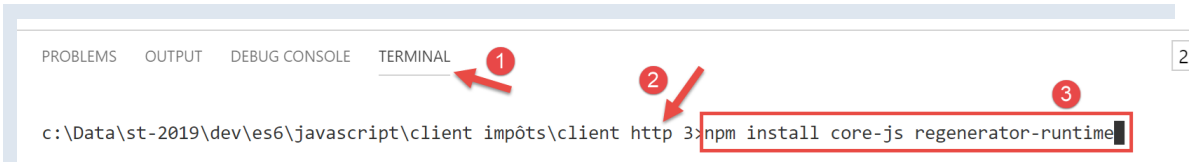
```
{
  "name": "client-http-3",
  "version": "1.0.0",
  "description": "client jS du serveur de calcul de l'impôt",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "serge.tahe@gmail.com",
  "license": "ISC",
  "devDependencies": {
    "@babel/core": "^7.6.0",
    "@babel/preset-env": "^7.6.0",
    "@webpack-cli/init": "^0.2.2",
    "babel-loader": "^8.0.6",
    "cross-env": "^6.0.0",
    "webpack": "^4.40.2",
    "webpack-cli": "^3.3.9",
    "webpack-dev-server": "^3.8.1"
  }
}
```

- lignes 12-19 : les dépendances du projet sont des **[devDependencies]** : on en a besoin pendant la phase de développement mais plus dans la phase de production. En effet, en production, c'est le fichier **[dist/main.js]** qui est utilisé. Il est codé en ES5 et n'a plus besoin des outils de transpilation de code ES6 vers du code ES5 ;

Il nous faut ajouter deux dépendances au projet :

- **[core-js]** : contient des « polyfills » pour ECMAScript 2019. Un polyfill permet d'exécuter un code récent, comme ECMAScript 2019 (sept 2019), sur des navigateurs anciens ;
- **[regenerator-runtime]** : selon le site de la bibliothèque --> **[Source transformer enabling ECMAScript 6 generator functions in JavaScript-of-today]** ;

Ces deux dépendances remplacent, à partir de Babel 7, la dépendance **[@babel/polyfill]** qui jouait auparavant ce rôle et qui est maintenant (sept 2019) dépréciée. Elles sont installées de la façon suivante :



Le fichier **[package.json]** évolue alors de la façon suivante :

```
1   {
2     "name": "client-http-3",
3     "version": "1.0.0",
4     "description": "My webpack project",
5     "main": "index.js",
6     "scripts": {
7       "test": "echo \"Error: no test specified\" && exit 1",
8       "build": "webpack",
9       "start": "webpack-dev-server"
10    },
11    "author": "serge.tahe@gmail.com",
12    "license": "ISC",
13    "devDependencies": {
14      "@babel/core": "^7.6.0",
15      "@babel/preset-env": "^7.6.0",
16      "@webpack-cli/init": "^0.2.2",
17      "babel-loader": "^8.0.6",
18      "babel-plugin-syntax-dynamic-import": "^6.18.0",
19      "html-webpack-plugin": "^3.2.0",
20      "webpack": "^4.40.2",
21      "webpack-cli": "^3.3.9",
22      "webpack-dev-server": "^3.8.1"
23    },
24    "dependencies": {
25      "core-js": "^3.2.1",
26      "regenerator-runtime": "^0.13.3"
27    }
28  }
```

L'utilisation des dépendances **[core-js, regenerator-runtime]** impose de mettre les **[imports]** suivants (lignes 3-4) dans le script principal **[src/main.js]** :

```
1   // imports
2   import axios from 'axios';
3   import "core-js/stable";
4   import "regenerator-runtime/runtime";
5
6   // imports
7   import Dao from './Dao2';
8   import Métier from './Métier';
```

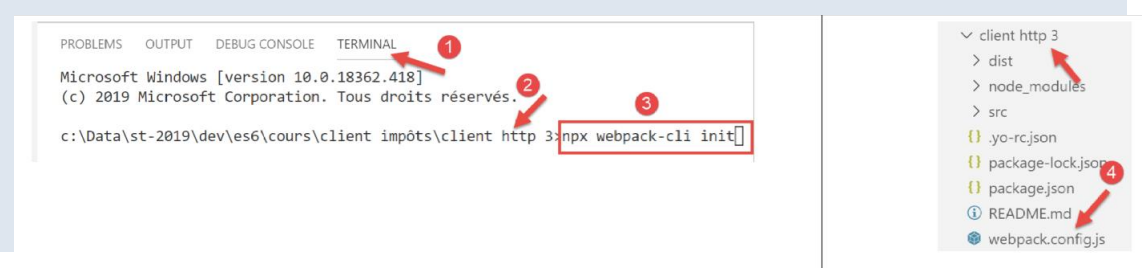
2.14.4.3 Configuration de [webpack]

[webpack] est l'outil qui va piloter :

- la transpilation ES6 → ES5 de tous les fichiers Javascript du projet ;
- l'assemblage des fichiers générés dans un unique fichier ;

Cet outil est piloté par un fichier de configuration **[webpack.config.js]** qui peut être généré grâce à une dépendance nommée **[@webpack-cli/init]** (sept 2019). Celle-ci a été installée avec les autres au paragraphe lien.

Nous exécutons la commande **[npx webpack-cli init]** dans un terminal **[VSCode]** :



Après avoir répondu aux différentes questions (dont on peut accepter la plupart des réponses proposées par défaut), un fichier **[webpack.config.js]** est généré à la racine du projet **[4]** :

Le fichier **[webpack.config.js]** ressemble à ceci :

```
1   /* eslint-disable */
2
3   const path = require('path');
4   const webpack = require('webpack');
5
6   /*
7    * SplitChunksPlugin is enabled by default and replaced
8    * deprecated CommonsChunkPlugin. It automatically identifies modules which
9    * should be splitted of chunk by heuristics using module duplication count and
10   * module category (i. e. node_modules). And splits the chunks…
11   *
12   * It is safe to remove "splitChunks" from the generated configuration
13   * and was added as an educational example.
14   *
15   * https://webpack.js.org/plugins/split-chunks-plugin/
16   *
17   */
18
19  const HtmlWebpackPlugin = require('html-webpack-plugin');
20
21  /*
22   * We've enabled HtmlWebpackPlugin for you! This generates a html
23   * page for you when you compile webpack, which will make you start
24   * developing and prototyping faster.
25   *
26   * https://github.com/jantimon/html-webpack-plugin
27   *
28   */
29
30  module.exports = {
31          mode: 'development',
32          entry: './src/index.js',
33
34          output: {
35                  filename: '[name].[chunkhash].js',
36                  path: path.resolve(__dirname, 'dist')
37          },
38
39          plugins: [new webpack.ProgressPlugin(), new HtmlWebpackPlugin()],
40
41          module: {
42                  rules: [
43                          {
44                                  test: /.(js|jsx)$/,
45                                  include: [path.resolve(__dirname, 'src')],
46                                  loader: 'babel-loader',
47
48                                  options: {
49                                          plugins: ['syntax-dynamic-import'],
50
51                                          presets: [
52                                                  [
```

```
53                                                      '@babel/preset-env',
54                                                      {
55                                                              modules: false
56                                                      }
57                                              ]
58                                      ]
59                              }
60                      }
61              ]
62      },
63
64      optimization: {
65              splitChunks: {
66                      cacheGroups: {
67                              vendors: {
68                                      priority: -10,
69                                      test: /[\\/]node_modules[\\/]/
70                              }
71                      },
72
73                      chunks: 'async',
74                      minChunks: 1,
75                      minSize: 30000,
76                      name: true
77              }
78      },
79
80      devServer: {
81              open: true
82      }
83  };
```

Je ne comprends pas tous les détails de ce fichier mais on peut remarquer quelques points :

- ligne 1 : le fichier ne contient pas du code ES6. **[Eslint]** déclare alors des erreurs qui remontent jusqu'à la racine du projet **[javascript]**. C'est gênant. Pour éviter qu'Eslint n'analyse un fichier, il suffit de mettre le commentaire de la ligne 1 ;
- ligne 31 : on travaille en mode **[développement]** ;
- ligne 32 : le script d'entrée est ici **[src/index.js]**. Nous serons amenés à changer cela ;
- ligne 36 : le dossier où seront tockées les produits de **[webpack]** sera le dossier **[dist]** ;
- ligne 46 : on voit que **[webpack]** utilise **[babel-loader]**, une des dépendances que nous avons installées ;
- ligne 54 : on voit que **[webpack]** utilise **[@babel-preset/env]**, une des dépendances que nous avons installées ;

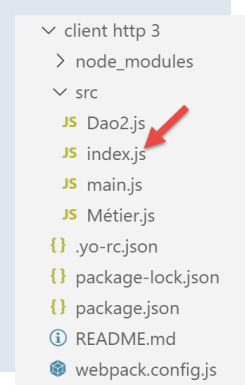
L'initialisation de **[webpack]** a modifié le fichier **[package.json]** (il demande l'autorisation) :

```
1   {
2     "name": "client-http-3",
3     "version": "1.0.0",
4     "description": "My webpack project",
5     "main": "index.js",
6     "scripts": {
7       "test": "echo \"Error: no test specified\" && exit 1",
8       "build": "webpack",
9       "start": "webpack-dev-server"
10    },
11    "author": "serge.tahe@gmail.com",
12    "license": "ISC",
13    "devDependencies": {
14      "@babel/core": "^7.6.0",
15      "@babel/preset-env": "^7.6.0",
16      "@webpack-cli/init": "^0.2.2",
17      "babel-loader": "^8.0.6",
18      "babel-plugin-syntax-dynamic-import": "^6.18.0",
19      "html-webpack-plugin": "^3.2.0",
20      "webpack": "^4.40.2",
21      "webpack-cli": "^3.3.9",
22      "webpack-dev-server": "^3.8.1"
23    },
24    "dependencies": {
25      "core-js": "^3.2.1",
26      "regenerator-runtime": "^0.13.3"
27    }
28  }
```

- ligne 4 : elle a été modifiée ;

- lignes 8-9, 18-19 : elles ont été ajoutées ;
- ligne 8 : la tâche **[npm]** qui permet de compiler le projet ;
- ligne 9 : la tâche **[npm]** qui permet de l'exécuter ;
- ligne 18 : ?
- ligne 19 : permet la génération d'un fichier **[dist/index.html]** embarquant automatiquement le script **[dist/main.js]** généré par **[webpack]** et c'est celui-ci qui est exploité lorsque le projet est exécuté ;

Enfin la configuration de **[webpack]** a généré un fichier **[src/index.js]** :



Le contenu de **[index.js]** est le suivant (sept 2019) :

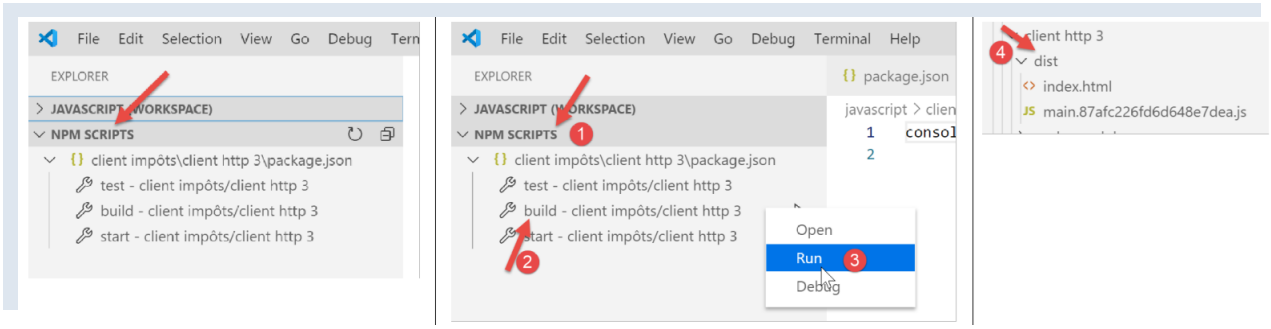
```
console.log("Hello World from your main file!");
```

2.14.4.4 Compilation et exécution du projet

Le fichier **[package.json]** a trois tâches **[npm]** :

```
1   "scripts": {
2       "test": "echo \"Error: no test specified\" && exit 1",
3       "build": "webpack",
4       "start": "webpack-dev-server"
5   },
```

Ces tâches sont comprises par **[VSCode]** qui les propose à l'exécution :



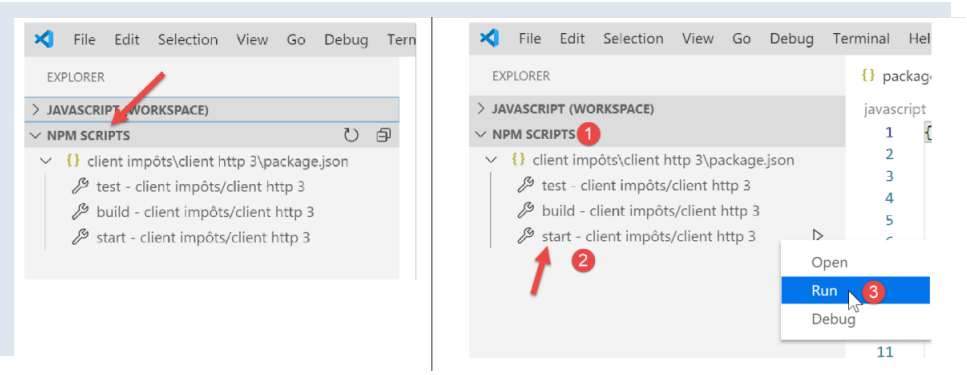
- en **[1-3]**, on compile le projet ;
- en **[4]** : le projet est compilé dans **[dist/main.hash.js]** et une page **[dist/index.html]** est créée ;

La page **[index.html]** générée est la suivante :

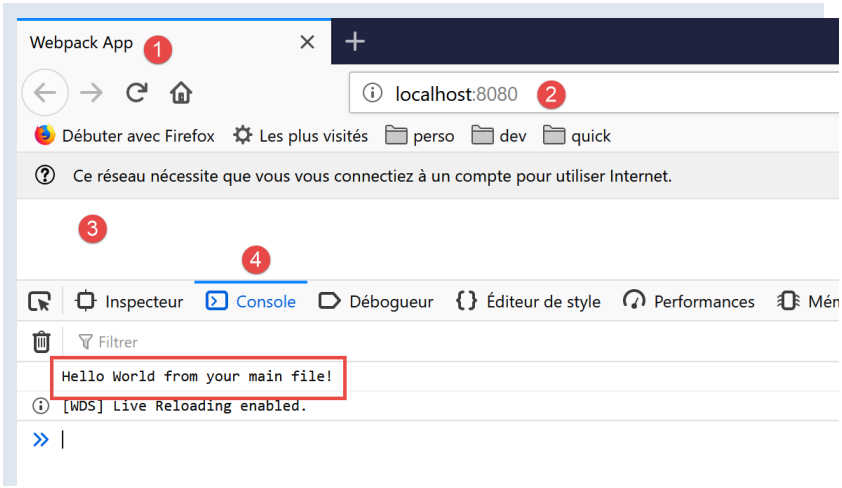
```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="UTF-8">
5       <title>Webpack App</title>
6     </head>
7     <body>
8     <script type="text/javascript" src="main.87afc226fd6d648e7dea.js"></script></body>
9   </html>
```

Cette page se contente donc d'encapsuler le fichier **[main.hash.js]** généré par **[webpack]**.

Le projet est exécuté par la tâche **[start]** :



La page **[dist/index.html]** est alors chargée sur un serveur, appartenant à la suite **[webpack]**, opérant sur le port 8080 de la machine locale et affichée par le navigateur par défaut de la machine :



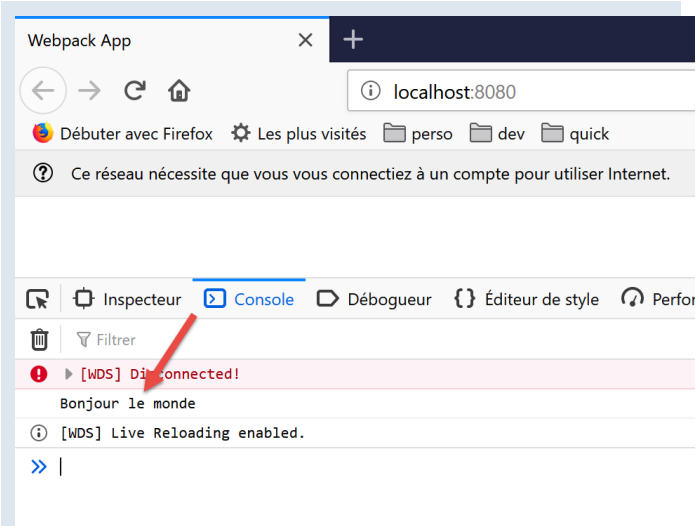
- en **[2]**, le port de service du serveur web de **[webpack]** ;
- en **[3]**, le corps de la page **[dist/index.html]** est vide ;
- en **[4]**, l'onglet **[console]** des outils de développement du navigateur, ici Firefox (F12) ;
- en **[5]**, le résultat de l'exécution du fichier **[src/index.js]**. On rappelle que le contenu de celui-ci était le suivant :

```
console.log("Hello World from your main file!");
```

Maintenant, changeons ce contenu en la ligne suivante :

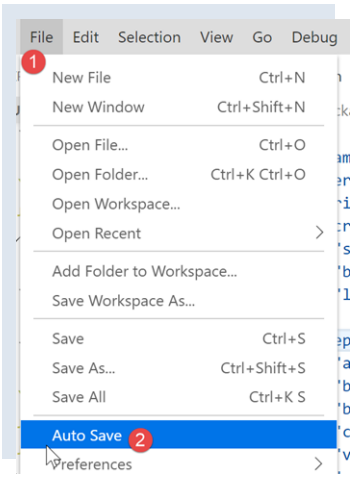
```
console.log("Bonjour le monde");
```

Automatiquement (sans recompiler), de nouveaux fichiers **[main.js, index.html]** sont générés et le nouveau fichier **[index.html]** chargé dans le navigateur :



Il n'est pas nécessaire d'exécuter la tâche **[build]** avant la tâche **[start]** : cette dernière fait d'abord la compilation du projet. Elle ne stocke pas les produits de cette compilation dans le dossier **[dist]**. Pour s'en apercevoir, il suffit de supprimer ce dossier. On verra alors que la tâche **[start]** compile et exécute le projet sans créer le dossier **[dist]**. Elle semble stocker ses produits **[index.html, main.hash.js]** dans un dossier propre à **[webpackdev-server]**. Ce comportement est suffisant pour nos tests.

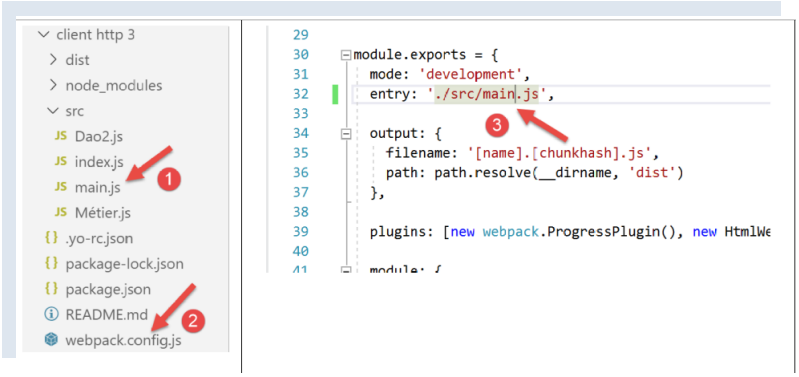
Lorsque le serveur de développement est lancé, toute modification sauvegardée d'un des fichiers du projet provoque une recompilation. Pour cette raison, nous inhibons le mode **[Auto Save]** de **[VSCode]**. En effet, nous ne voulons pas de recompilation dès qu'on tape des caractères dans un des fichiers du projet. Nous ne voulons de recompilation qu'au moment des sauvegardes des modifications :



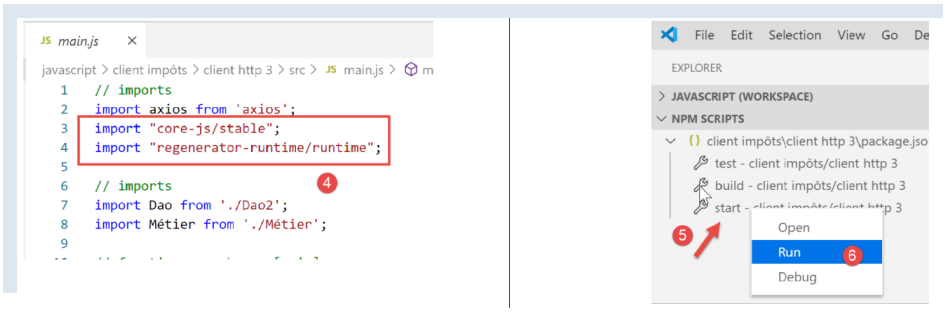
- en **[2]**, l'option **[Auto Save]** ne doit pas être cochée ;

2.14.4.5 Tests du client Javascript du serveur de calcul de l'impôt

Pour tester le client Javascript du serveur de calcul de l'impôt, il faut désigner **[main.js] [1]** comme le point d'entrée du projet dans le fichier **[webpack.config.js] [2-3]** :



N'oublions pas que le script **[main.js]** doit inclure deux imports supplémentaires par rapport à sa version dans **[Client http 2]** :



Par ailleurs, nous avons légèrement modifié le code pour gérer les erreurs que peut envoyer le serveur :

```
// imports
import axios from 'axios';
import "core-js/stable";
import "regenerator-runtime/runtime";

// imports
import Dao from './Dao2';
import Métier from './Métier';

// fonction asynchrone [main]
async function main() {
  // configuration axios
  axios.defaults.timeout = 2000;
  axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
  // instanciation couche [dao]
  const dao = new Dao(axios);
  // requêtes HTTP
  let taxAdminData;
  try {
    // init session
    log("----------init-session");
    let response = await dao.initSession();
    log(response);
    if (response.état != 700) {
      throw new Error(JSON.stringify(response.réponse));
    }
    // authentification
    log("----------authentifier-utilisateur");
    response = await dao.authentifierUtilisateur("admin", "admin");
    log(response);
    if (response.état != 200) {
      throw new Error(JSON.stringify(response.réponse));
    }
    // données fiscales
    log("----------get-admindata");
    response = await dao.getAdminData();
    log(response);
    if (response.état != 1000) {
      throw new Error(JSON.stringify(response.réponse));
    }
    taxAdminData = response.réponse;
  } catch (error) {
    // on logue l'erreur
```

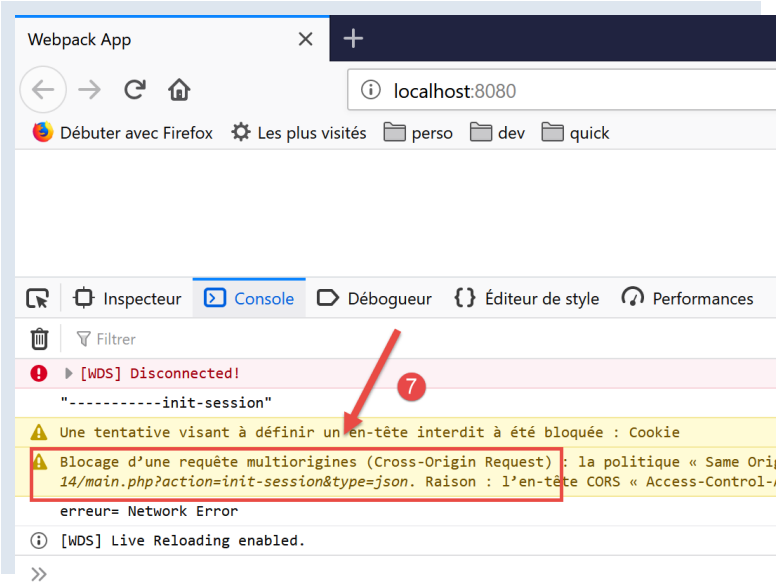
```
44      console.log("erreur=", error.message);
45      // fin
46      return;
47    }
48
49    // instanciation couche [métier]
50    const métier = new Métier(taxAdminData);
51
52    // calculs d'impôt
53    log("-----------calculer-impot x 3");
54    const simulations = [];
55    simulations.push(métier.calculerImpot("oui", 2, 45000));
56    simulations.push(métier.calculerImpot("non", 2, 45000));
57    simulations.push(métier.calculerImpot("non", 1, 30000));
58    // liste des simulations
59    log("-----------liste-des-simulations");
60    log(simulations);
61    // suppression d'une simulation
62    log("-----------suppression simulation n° 1");
63    simulations.splice(1, 1);
64    log(simulations);
65  }
66
67  // log jSON
68  function log(object) {
69    console.log(JSON.stringify(object, null, 2));
70  }
71
72  // exécution
73  main();
```

Commentaires

- aux lignes **[24-26]**, **[31-33]**, **[38-40]**, on teste le code **[response.état]** envoyé dans la réponse jSON du serveur. Si ce code dénote une erreur, une exception est lancée avec pour message d'erreur la chaîne jSON de la réponse du serveur **[response.réponse]** ;

Ceci fait, nous exécutons le projet **[5-6]**.

La page **[index.html]** est alors générée et chargée dans le navigateur :



- en **[7]**, on voit que l'action **[init-session]** n'a pu aller à son terme à cause d'un problème **[CORS]** (Cross-Origin Resource Sharing) ;

Le problème CORS vient de la relation client / serveur :

- notre client Javascript a été téléchargée sur la machine **[http://localhost:8080]**;
- le serveur de calcul d'impôt s'exécute sur la machine **[http://localhost:80]**;

- le client et le serveur ne sont pas alors dans les mêmes domaines (même machine mais pas même port);
- le navigateur qui exécute le client Javascript chargé à partir de la machine **[http://localhost:8080]** bloque toute requête qui n'a pas pour cible **[http://localhost:80]**. C'est une mesure de sécurité. Aussi bloque-t-il la requête du client vers le serveur qui opère sur la machine **[http://localhost:80]**;

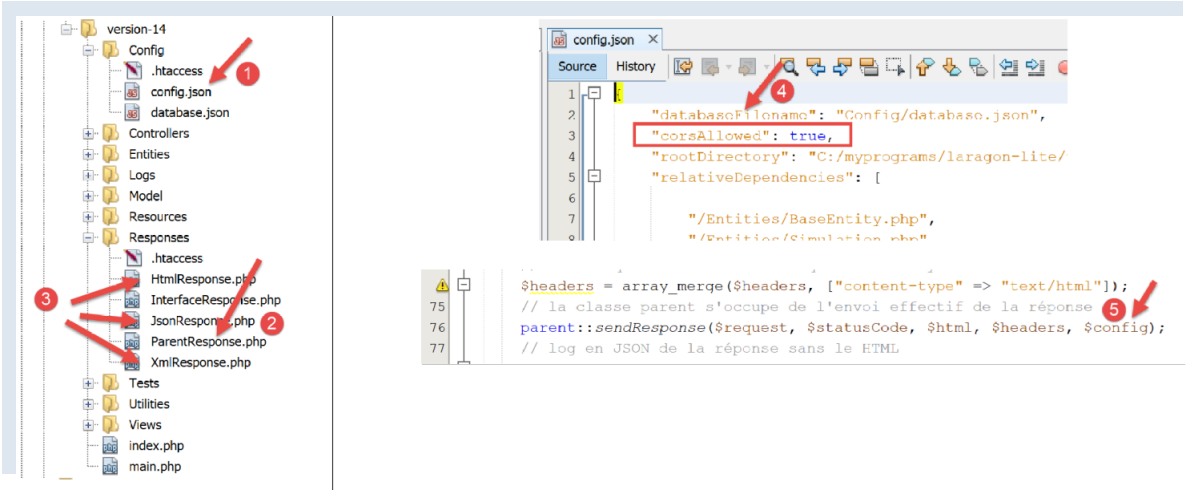
En fait, le navigateur ne bloque pas totalement la requête. Il attend en fait que le serveur lui 'dise' qu'il accepte les requêtes inter-domaines. S'il obtient cette autorisation, le navigateur transmettra alors la requête inter-domaines.

Le serveur donne son autorisation en envoyant des entêtes HTTP particuliers :

```
1   Access-Control-Allow-Origin: http://localhost:8080
2   Access-Control-Allow-Headers: Accept, Content-Type
3   Access-Control-Allow-Methods: GET, POST
4   Access-Control-Allow-Credentials: true
```

- ligne 1 : le client Javascript opère sur le domaine **[http://localhost:8080]**. Le serveur doit explicitement répondre qu'il accepte ce domaine ;
- ligne 2 : le client Javascript va utiliser dans ses requêtes les entêtes HTTP **[Accept, Content-Type]** :
  - **[Accept]** : cet entête est envoyé dans toute requête ;
  - **[Content-Type]** : cet entête est utilisé dans les opérations POST pour indiquer le type des paramètres du POST ;
    - Le serveur doit explicitement accepter ces deux entêtes HTTP ;
- ligne 3 : le client Javascript va utiliser des requêtes GET et POST. Le serveur doit explicitement accepter ces deux types de requêtes ;
- ligne 4 : le client Javascript va envoyer des cookies de session. Le serveur les accepte avec l'entête de la ligne 4 ;

Il nous faut donc modifier le serveur. Nous faisons cela dans **[Netbeans]**. Le problème des CORS est un problème rencontré uniquement en mode développement. En production, le client et le serveur travailleront dans le même domaine **[http://localhost:80]** et il n'y aura pas de problème CORS. Il nous faut donc un moyen d'autoriser ou pas les requêtes CORS par configuration du serveur.



Les modifications du serveur se font à trois endroits :

- **[1, 4]** : dans le fichier de configuration **[config.json]** pour y mettre un booléen qui contrôlera l'acceptation ou non des requêtes inter-domaines ;
- **[2]** : dans la classe **[ParentResponse]** qui envoie la réponse au client Javascript. C'est elle qui enverra les entêtes CORS attendus par le navigateur client ;
- **[3]** : dans les classes **[HtmlResponse, JsonResponse, XmlResponse]** qui génèrent les réponses pour respectivement les sessions **[html, json, xml]**. Ces classes doivent passer à leur classe parente **[2]**, le booléen **[corsAllowed]** trouvé en **[4]**. Cela se fait en **[5]**, en passant le tableau image du fichier jSON **[2]** ;

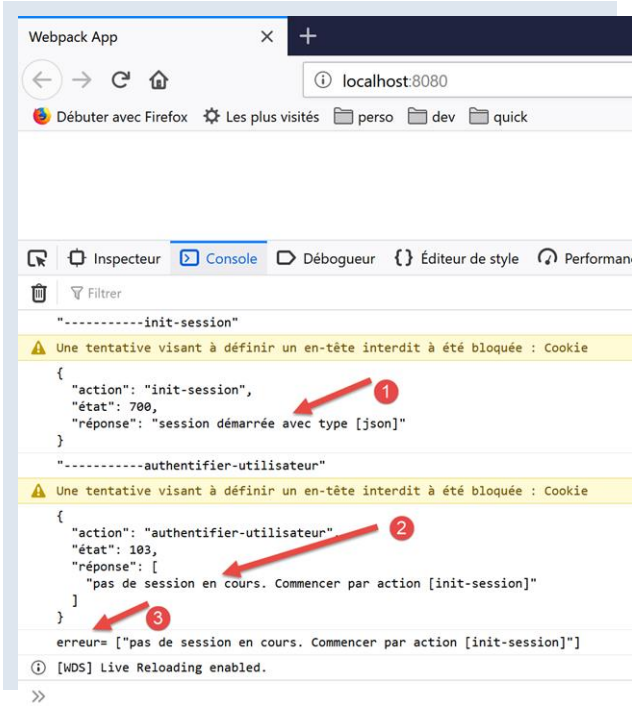
La classe **[ParentResponse] [2]** évolue de la façon suivante :

```
1   <?php
2
3   namespace Application;
4
5   // dépendances Symfony
```

```
6   use Symfony\Component\HttpFoundation\Response;
7   use Symfony\Component\HttpFoundation\Request;
8
9   class ParentResponse {
10
11    // int $statusCode : le code HTTP de statut de la réponse
12    // string $content : le corps de la réponse à envoyer
13    // selon les cas, c'est une chaîne JSON, XML, HTML
14    // array $headers : les entêtes HTTP à ajouter à la réponse
15
16    public function sendResponse(
17      Request $request,
18      int $statusCode,
19      string $content,
20      array $headers,
21      array $config): void {
22
23      // préparation de la réponse texte du serveur
24      $response = new Response();
25      $response->setCharset("utf-8");
26      // code de statut
27      $response->setStatusCode($statusCode);
28      // headers pour les requêtes inter-domaines
29      if ($config['corsAllowed']) {
30        $origin = $request->headers->get("origin");
31        if (strpos($origin, "http://localhost") === 0) {
32          $headers = array_merge($headers,
33            ["Access-Control-Allow-Origin" => $origin,
34              "Access-Control-Allow-Headers" => "Accept, Content-Type",
35              "Access-Control-Allow-Methods" => "GET, POST",
36              "Access-Control-Allow-Credentials" => "true"
37          ]);
38        }
39      }
40      foreach ($headers as $text => $value) {
41        $response->headers->set($text, $value);
42      }
43      // cas particulier de la méthode [OPTIONS]
44      // seuls les entêtes sont importants dans ce cas
45      $method = strtolower($request->getMethod());
46      if ($method === "options") {
47        $content = "";
48        $response->setStatusCode(Response::HTTP_OK);
49      }
50      // on envoie la réponse
51      $response->setContent($content);
52      $response->send();
53    }
54
55  }
```

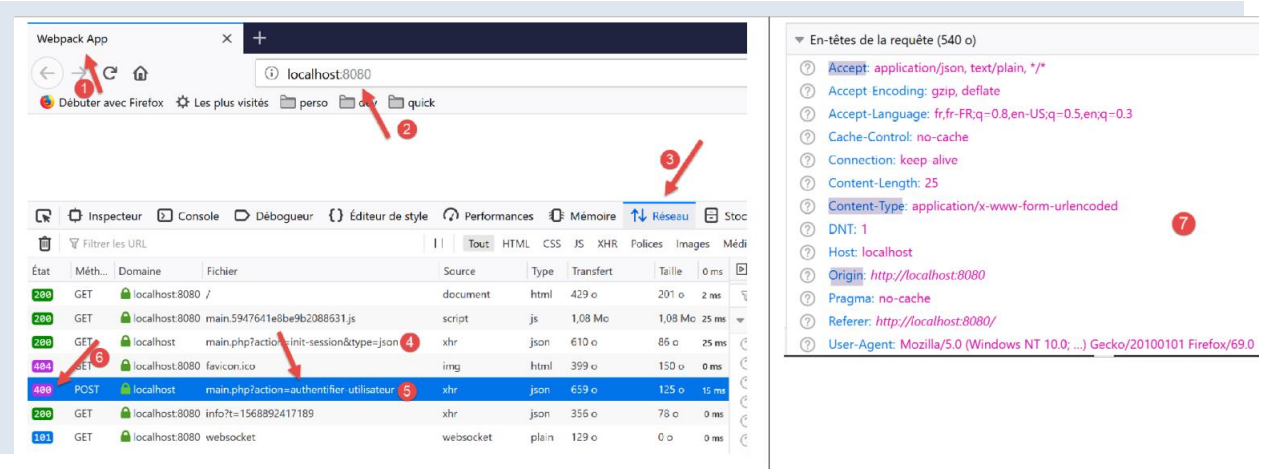
- ligne 29 : on regarde si on doit gérer les requêtes inter-domaines. Si oui, on va générer les entêtes HTTP CORS (lignes 33-37) même si la requête courante n'est pas une requête inter-domaines. Dans ce dernier cas, les entêtes CORS seront inutiles et resteront inexploités par le client ;
- ligne 30 : dans une requête inter-domaines, le navigateur client qui interroge le serveur envoie un entête HTTP **[Origin: http://localhost:8080]** (dans le cas précis de notre client Javascript). Ligne 30, on récupère cet entête HTTP dans la requête **[$request]** ;
- ligne 31 : on n'acceptera des requêtes inter-domaines provenant uniquement de la machine **[http://localhost]**. On rappelle que ces requêtes n'ont lieu qu'en mode développement du projet ;
- lignes 32-36 : on ajoute les entêtes CORS aux entêtes déjà présents dans le tableau **[$headers]** ;
- lignes 45-49 : la façon dont le navigateur client demande les autorisations CORS peut différer selon le client exécuté. Il arrive parfois que le navigateur client demande ces autorisations avec une commande HTTP **[OPTIONS]**. C'est une nouveauté pour notre serveur qui a été construit pour servir uniquement les commandes **[GET, POST]**. Dans le cas d'une commande **[OPTIONS]**, le serveur génère actuellement une réponse d'erreur. Lignes 46-49, nous corrigeons cela au dernier moment : si ligne 46, nous constatons que la commande courante est une commande **[OPTIONS]**, alors on génère pour le client :
  - lignes 47, 51 : une réponse **[$content]** vide ;
  - ligne 48 : un code de statut de 200 indiquant que la commande est réussie. La seule chose importante pour cette commande est l'envoi des entêtes CORS des lignes 33-36. C'est ce qu'attend le navigateur client ;

Une fois le serveur ainsi corrigé, le client Javascript s'exécute mieux mais fait apparaître une nouvelle erreur :



- en **[1]**, la session jSON est correctement initialisée ;
- en **[2]**, l'action **[authentifier-utilisateur]** échoue : le serveur indique qu'il n'y a pas de session en cours. Cela signifie que le client Javascript ne lui pas renvoyé correctement le cookie de session qu'il a envoyé lors de l'action **[init-session]** ;

Examinons les échanges réseau qui ont eu lieu :



- en **[4]**, la requête **[init-session]**. Elle s'est bien déroulée avec un code 200 pour le statut de la réponse ;
- en **[5]**, la requête **[authentifier-utilisateur]**. Celle-ci échoue avec un code 400 (Bad Request) **[6]** pour le statut de la réponse ;

Si on examine les entêtes HTTP **[7]** de la requête **[5]**, on peut voir que le client Javascript n'a pas envoyé l'entête HTTP **[Cookie]** qui lui aurait permis de renvoyer le cookie de session envoyé initialement par le serveur. C'est la raison pour laquelle celui-ci déclare qu'il n'y a pas de session.

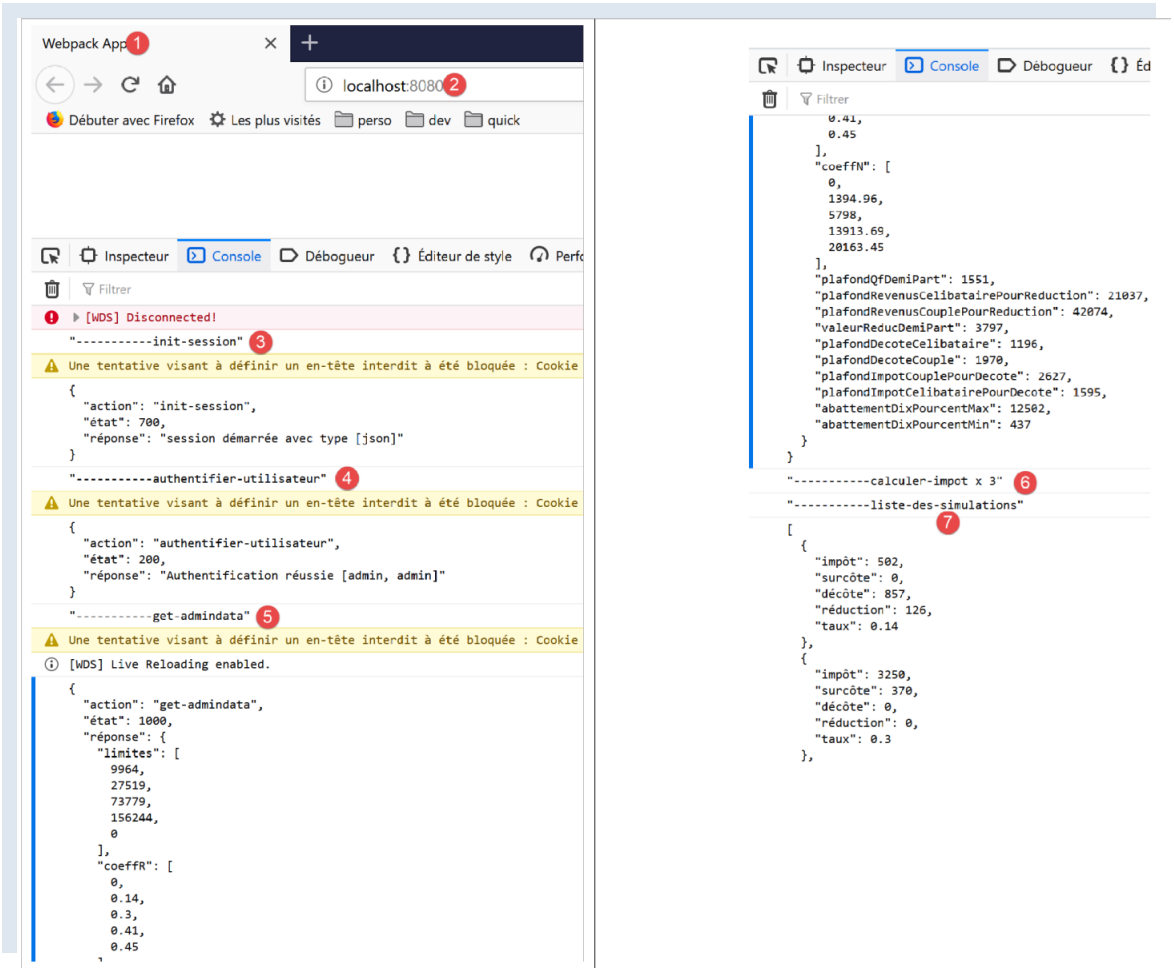
Pour que le client envoie le cookie de session, il faut ajouter une configuration à l'objet **[axios]** :

```
1   // imports
2   import axios from 'axios';
3   import "core-js/stable";
4   import "regenerator-runtime/runtime";
5
6   // imports
7   import Dao from './Dao2';
8   import Métier from './Métier';
```

```
9
10  // fonction asynchrone [main]
11  async function main() {
12    // configuration axios
13    axios.defaults.timeout = 2000;
14    axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
15    axios.defaults.withCredentials = true;
16    // instanciation couche [dao]
17    const dao = new Dao(axios);
18    // requêtes HTTP
19    let taxAdminData;
20  ...
```

La ligne 15 demande à ce que les cookies soient inclus dans les entêtes HTTP de la requête **[axios]**. Remarquons que cela n'avait pas été nécessaire dans l'environnement **[node.js]**. Il y a donc des différences de code entre les deux environnements.

Une fois cette erreur corrigée, le client Javascript se déroule normalement :



2.14.5 Amélioration du client HTTP 3

Lorsque la classe **[Dao2]** précédente s'exécute au sein d'un navigateur, la gestion du cookie de session est inutile. En effet, c'est le navigateur qui héberge la couche **[dao]** qui gère le cookie de session : il renvoie automatiquement tout cookie que le serveur lui envoie. Du coup la classe **[Dao2]** peut être réécrite en la classe **[Dao3]** suivante :

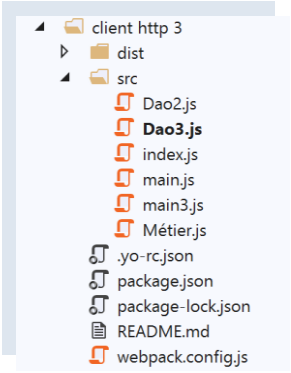
```
1   "use strict";
2
3   // imports
4   import qs from "qs";
5
6   class Dao3 {
7     // constructeur
8     constructor(axios) {
```

```
9       this.axios = axios;
10    }
11
12    // init session
13    async initSession() {
14      // options de la requête HHTP [get /main.php?action=init-session&type=json]
15      const options = {
16        method: "GET",
17        // paramètres de l'URL
18        params: {
19          action: "init-session",
20          type: "json"
21        }
22      };
23      // exécution de la requête HTTP
24      return await this.getRemoteData(options);
25    }
26
27    async authentifierUtilisateur(user, password) {
28      // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
29      const options = {
30        method: "POST",
31        headers: {
32          "Content-type": "application/x-www-form-urlencoded"
33        },
34        // corps du POST
35        data: qs.stringify({
36          user: user,
37          password: password
38        }),
39        // paramètres de l'URL
40        params: {
41          action: "authentifier-utilisateur"
42        }
43      };
44      // exécution de la requête HTTP
45      return await this.getRemoteData(options);
46    }
47
48    async getAdminData() {
49      // options de la requête HHTP  [get /main.php?action=get-admindata]
50      const options = {
51        method: "GET",
52        // paramètres de l'URL
53        params: {
54          action: "get-admindata"
55        }
56      };
57      // exécution de la requête HTTP
58      const data = await this.getRemoteData(options);
59      // résultat
60      return data;
61    }
62
63    async getRemoteData(options) {
64      // exécution de la requête HTTP
65      let response;
66      try {
67        // requête asynchrone
68        response = await this.axios.request("main.php", options);
69      } catch (error) {
70        // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
71        if (error.response) {
72          // la réponse du serveur est dans [error.response]
73          response = error.response;
74        } else {
75          // on relance l'erreur
76          throw error;
77        }
78      }
79      // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
80      // la réponse du serveur est dans [response.data]
81      return response.data;
82    }
83  }
84
85  // export de la classe
```

```
86  export default Dao3;
```

Tout ce qui avait trait à la gestion du cookie de gestion a disparu.

Nous modifions le projet précédent de la façon suivante :



Dans le dossier **[src]**, nous avons ajouté deux fichiers :

- la classe **[Dao3]** que nous venons de présenter ;
- le fichier **[main3]** chargée de lancer la nouvelle version ;

Le fichier **[main3]** reste identique au fichier **[main]** de la version précédente mais il utilise désormais la classe **[Dao3]** :

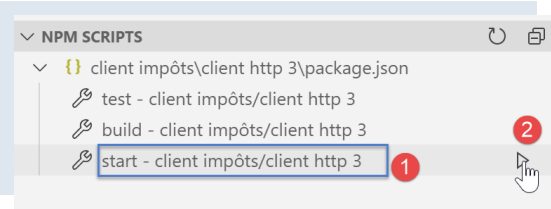
```
1   // imports
2   import axios from "axios";
3   import "core-js/stable";
4   import "regenerator-runtime/runtime";
5
6   // imports
7   import Dao from "./Dao3";
8   import Métier from "./Métier";
9
10  // fonction asynchrone [main]
11  async function main() {
12    // configuration axios
13    axios.defaults.timeout = 2000;
14    axios.defaults.baseURL =
15      "http://localhost/php7/scripts-web/impots/version-14";
16    axios.defaults.withCredentials = true;
17    // instanciation couche [dao]
18    const dao = new Dao(axios);
19    // requêtes HTTP
20    ...
21  }
22
23  // log jSON
24  function log(object) {
25    console.log(JSON.stringify(object, null, 2));
26  }
27
28  // exécution
29  main();
```

Le fichier **[webpack.config]** est modifié pour exécuter maintenant, le script **[main3]** :

```
1   /* eslint-disable */
2
3   const path = require("path");
4   const webpack = require("webpack");
5
6   /*
7    * SplitChunksPlugin is enabled by default and replaced
8    * deprecated CommonsChunkPlugin. It automatically identifies modules which
9    * should be splitted of chunk by heuristics using module duplication count and
10   * module category (i. e. node_modules). And splits the chunks…
11   *
```

```
 * It is safe to remove "splitChunks" from the generated configuration
 * and was added as an educational example.
 *
 * https://webpack.js.org/plugins/split-chunks-plugin/
 *
 */

const HtmlWebpackPlugin = require("html-webpack-plugin");

/*
 * We've enabled HtmlWebpackPlugin for you! This generates a html
 * page for you when you compile webpack, which will make you start
 * developing and prototyping faster.
 *
 * https://github.com/jantimon/html-webpack-plugin
 *
 */

module.exports = {
  mode: "development",
  //entry: "./src/mainjs",
  entry: "./src/main3.js",
  output: {
    filename: "[name].[chunkhash].js",
    path: path.resolve(__dirname, "dist")
  },

  plugins: [new webpack.ProgressPlugin(), new HtmlWebpackPlugin()],
...
};
```

Ceci fait, on exécute le projet après avoir lancé le serveur de calcul de l'impôt :



Les résultats obtenus dans la console du navigateur sont identiques à ceux de la version précédente.

2.14.6 Conclusion

Nous avons désormais désormais tous les outils pour développer le code Javascript d'une application web. Nous pouvons :

- utiliser le code ECMAScript le plus récent ;
- tester des éléments isolés de ce code dans un environnement **[node.js]** plus simple pour le débogage et les tests ;
- porter ensuite ce code dans un navigateur grâce aux outils **[babel]** et **[webpack]** ;

INTRODUCTION AU FRAMEWORK

VUE.JS

PAR L'EXEMPLE

Serge Tahé, octobre 2019

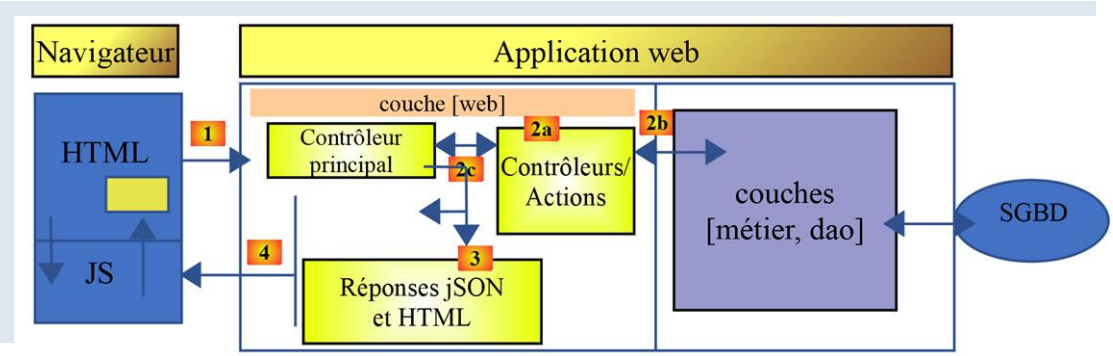
3 Introduction au framework VUE.JS par l'exemple

3.1 Présentation du cours

La dernière version du serveur de calcul de l'impôt développée dans le document **[1]** (cf paragraphe) peut être améliorée de diverses manières :

- la version écrite est centrée sur le serveur. La tendance est désormais (sept 2019) au client / serveur :
  - le serveur fonctionne en service jSON ;
  - une page statique ou non est le point d'entrée de l'application web. Cette page contient du HTML /CSS mais aussi du JavaScript ;
  - les autres pages de l'application web sont obtenues dynamiquement par le JavaScript :
    - la page HTML peut être obtenue par assemblage de fragments statiques ou non, fournis par le même serveur qui a fourni la page d'entrée, ou bien construites par le Javascript du client ;
    - ces différentes pages affichent des données qui sont demandées au service jSON ;

Ainsi le travail est réparti sur le client et le serveur. Le serveur ainsi déchargé peut servir davantage d'utilisateurs. L'architecture correspondant à ce modèle est le suivant :



JS : JavaScript

Le navigateur est client :

- d'un service de pages ou fragments statiques ou non (non représenté ci-dessus) ;
- d'un service de données jSON ;

Le code JavaScript est donc un client jSON et à ce titre peut être organisé en couches **[UI, métier, dao]** (UI : User Interface) comme l'ont été nos clients jSON écrits en PHP. Au final, le navigateur ne charge qu'une unique page, la page d'accueil. Toutes les autres sont obtenues et construites par le JavaScript. On appelle ce type d'application **SPA** : **S**ingle **P**age **A**pplication ou encore **APU** : **A**pplication à **P**age **U**nique.

Ce type d'application fait également partie des applications dites **AJAX** : **A**synchronous **J**avascript **A**nd **X**M :

- **Asynchronous** : parce que les appels du client Javascript au serveur jSON sont asynchrones ;
- **XML** : parce que XML était la technologie utilisée avant l'avènement du jSON. On a cependant gardé l'acronyme AJAX ;

Nous allons étudier une telle architecture dans ce document. Côté client, nous utiliserons le framework Javascript **[Vue.js]** **[https://vuejs.org/]** pour écrire le client Javascript du serveur jSON PHP que nous avons écrit dans le document **[1]**.

[Vue.js] est un framework JavaScript. Nous avons présenté le langage Javascript dans le document **[2]**. Nous ne ferons pas une étude exhaustive du framework **[vue.js]**. Nous nous contenterons de présenter les concepts qui seront utilisés ensuite dans l'écriture d'un client **[Vue.js]** pour la version jSON de la version 14 du serveur de calcul de l'impôt (cf |https://tahe.developpez.com/tutoriels-cours/php7|).

Les scripts de ce document sont commentés et leur exécution console reproduite. Des explications supplémentaires sont parfois fournies. Le document nécessite une lecture active : pour comprendre un script, il faut à la fois lire son code, ses commentaires et ses résultats d'exécution.

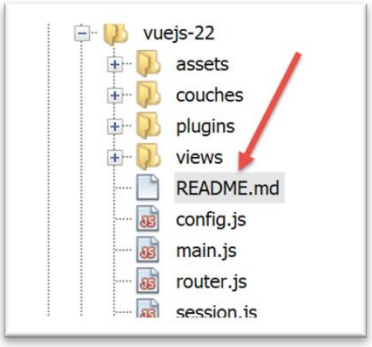
Les exemples du document sont disponibles |**ici**|.

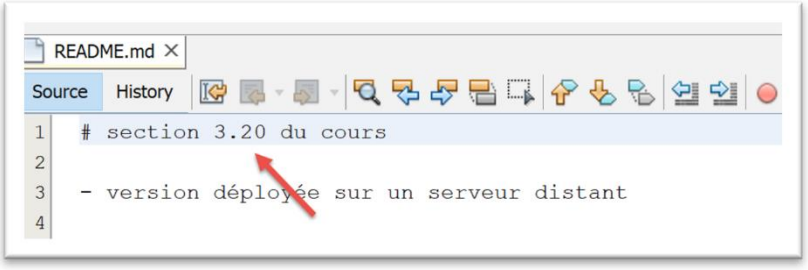
L'application serveur PHP 7 peut être testée |**ici**|.

Serge Tahé, octobre 2019

Le contenu de cette partie du cours est le suivant :

Certains lecteurs préfèreront lire, modifier et tester le code plutôt que de lire un cours. Dans chaque dossier des codes de ce document on a mis un fichier **[README.md]** résumant le contenu du dossier et faisant le lien avec le cours :



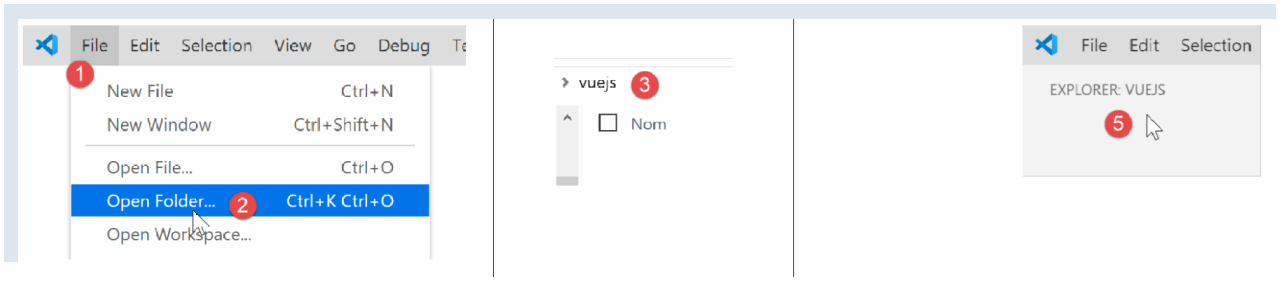


3.2 Création d'un environnement de travail

Nous reprenons l'environnement de travail détaillé dans le document **[2]** :

- Visual Studio Code (VSCode) pour écrire les codes JavaScript ;
- **[node.js]** pour les exécuter ;
- **[npm]** pour télécharger et installer les bibliothèques JavaScript dont nous aurons besoin ;

Nous créons un environnement de travail dans **[VSCode]** :

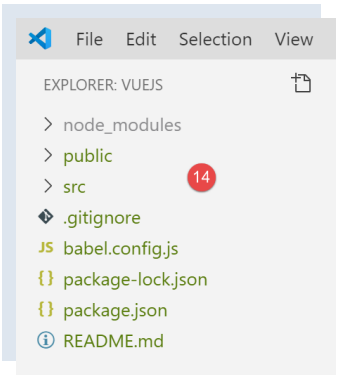


- en **[1-5]**, nous ouvrons un dossier **[vuejs]** vide dans **[VSCode]** ;

- en **[8-10]**, on installe la dépendance **[@vue/cli]** qui va nous permettre d'initialiser un projet **[vue.js]**. Cette dépendance amène un grand nombre de packages (plusieurs centaines) ;

Dans le même terminal, on tape ensuite la commande **[vue create .]** qui demande à créer un projet **[vue.js]** dans le dossier courant :

- en **[13]**, commence une série de questions qui servent à configurer le projet ;



- une fois toutes les questions répondues, de nouveaux packages sont téléchargés et un projet généré dans le dossier courant **[14]**.

Regardons ce qui a été généré. Le fichier **[package.json]** est le suivant :

```
1   {
2     "name": "vuejs",
3     "version": "0.1.0",
4     "private": true,
5     "scripts": {
6       "serve": "vue-cli-service serve vuejs-20/main.js",
7       "build": "vue-cli-service build vuejs-20/main.js",
8       "lint": "vue-cli-service lint"
9     },
10    "dependencies": {
11      "axios": "^0.19.0",
12      "bootstrap": "^4.3.1",
13      "bootstrap-vue": "^2.0.2",
14      "core-js": "^2.6.5",
15      "vue": "^2.6.10",
16      "vue-router": "^3.1.3",
17      "vuex": "^3.1.1"
18    },
19    "devDependencies": {
20      "@vue/cli-plugin-babel": "^3.11.0",
21      "@vue/cli-plugin-eslint": "^3.11.0",
22      "@vue/cli-service": "^3.11.0",
23      "babel-eslint": "^10.0.1",
24      "eslint": "^5.16.0",
25      "eslint-plugin-vue": "^5.0.0",
26      "vue-template-compiler": "^2.6.10"
27    },
28    "eslintConfig": {
29      "root": true,
30      "env": {
31        "node": true
32      },
33      "extends": [
34        "plugin:vue/essential",
35        "eslint:recommended"
36      ],
37      "rules": {},
38      "parserOptions": {
39        "parser": "babel-eslint"
40      }
41    },
42    "postcss": {
43      "plugins": {
44        "autoprefixer": {}
45      }
46    },
47    "browserslist": [
48      "> 1%",
49      "last 2 versions"
50    ]
51  }
```

Commentaires

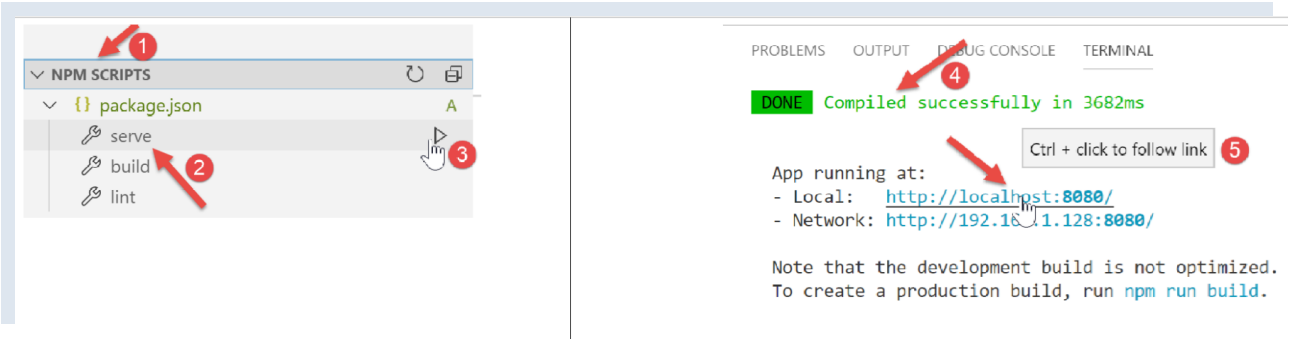
- lignes 14-22 : dans les dépendances nécessaires au développement on voit des références aux deux outils **[eslint, babel]** déjà utilisés dans les deux chapitres précédents. S'y ajoutent des plugins de ces deux outils destinés à leur utilisation au sein de **[vue.js]** ;
- ligne 34 : c'est le package **[babel-eslint]** qui opèrera la transpilation ES6 -> ES5 des codes jS ;
- lignes 5-9 : trois tâches **[npm]** ont été créées :
  - **[build]** : sert à construire la version compilée du projet prête à entrer en production ;
  - **[serve]** : exécute le projet sur un serveur web. C'est avec cet outil que sont faits les tests lors du développement. Comme avec **[webpack-dev-server]**, une modification d'un code source du projet provoque automatiquement la recompilation du projet et son rechargement par le serveur web ;
  - **[lint]** : sert à analyser les codes jS et délivre des rapports. Nous n'utiliserons pas cet outil ici ;

Un fichier **[README.md]** a été généré avec le contenu suivant :

```
# vuejs

## Project setup
```
npm install
```

### Compiles and hot-reloads for development
```
npm run serve
```

### Compiles and minifies for production
```
npm run build
```

### Run your tests
```
npm run test
```

### Lints and fixes files
```
npm run lint
```

### Customize configuration
See [Configuration Reference](https://cli.vuejs.org/config/).
```

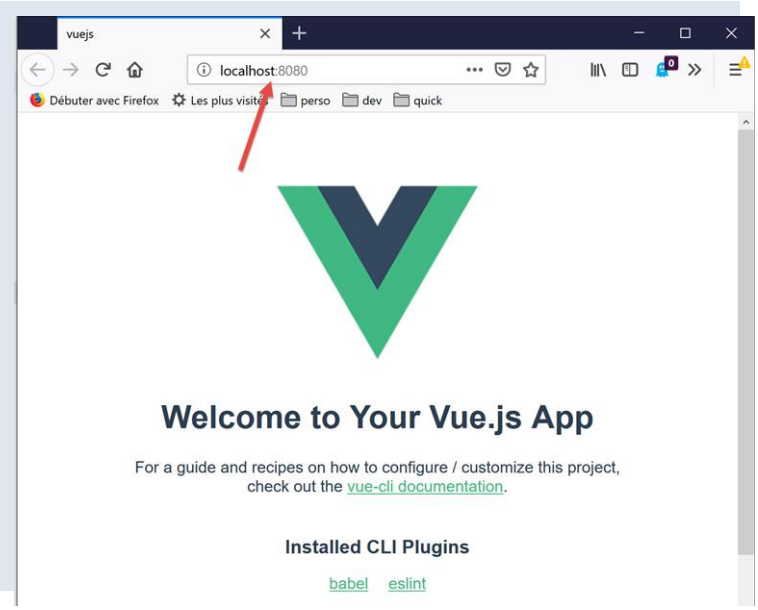
Ce fichier résume les commandes à utiliser pour gérer le projet.

Nous savons que dans **[VSCode]**, les tâches **[npm]** sont proposées à l'exécution :



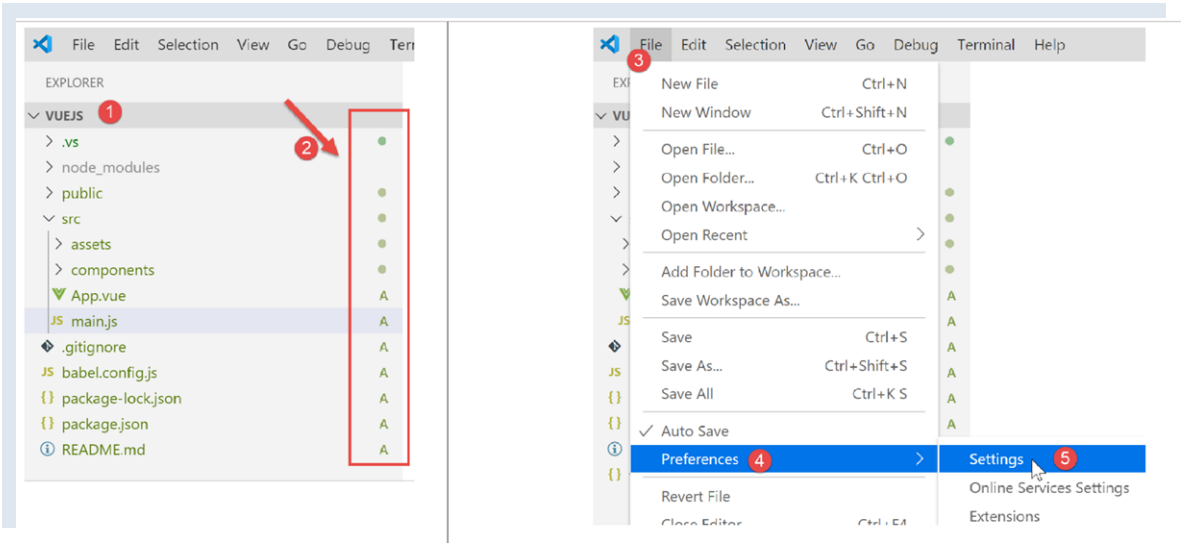
- en **[1-3]**, nous exécutons la commande **[serve]** qui va compiler, puis exécuter le projet **[4-5]** ;

A l'URL **[http://localhost:8080]**, nous obtenons la page suivante :

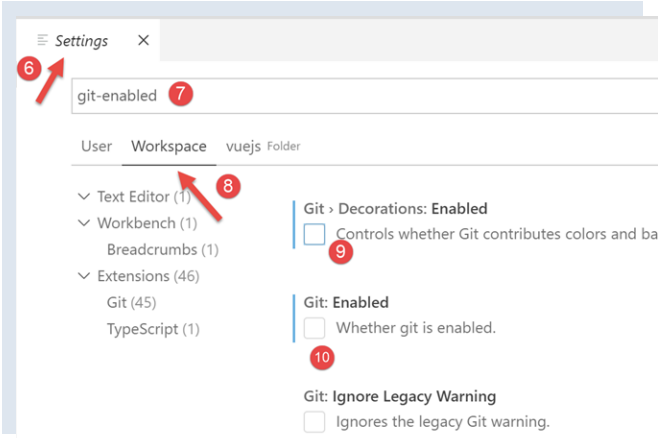


Nous expliquerons un peu plus loin ce qui a amené à cette page.

Continuons à configurer notre environnement de travail :



- en **[2]** ci-dessus, nous voyons des indicateurs **[git]**. **[git]** est un gestionnaire de code source permettant de gérer des versions successives de celui-ci et de les partager entre développeurs. Nous allons désactiver cet outil pour le projet ;
- en **[3-5]**, nous allons dans les propriétés du projet ;



- en **[9-10]**, on désactive l'utilisation de **[git]** dans le projet ;

Nous allons écrire divers tests pour montrer le fonctionnement de **[vue.js]**. Nous ne voulons cependant pas créer à chaque fois un nouveau projet car il faudrait alors à chaque fois générer un dossier **[node_modules]** alors que celui-ci fait plusieurs centaines de méga-octets. Revenons sur les tâches **[npm]** du fichier **[package.json]** :

```
1.    "scripts": {
2.      "serve": "vue-cli-service serve vuejs-00/main.js",
3.      "build": "vue-cli-service build vuejs-00/main.js",
4.      "lint": "vue-cli-service lint"
5.  },
```

- ligne 2 : la commande **[serve]** utilise par défaut :
  - le fichier **[public/index.html]** ;
  - associé au fichier **[src/main.js]** ;

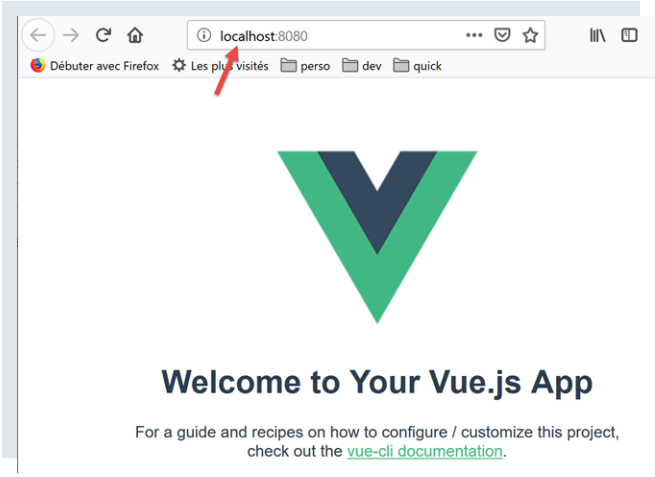
Ligne 2, il est possible de préciser à la commande **[serve]**, le point d'entrée du projet, par exemple :

```
"serve": "vue-cli-service serve vuejs-00/main.js",
```

Essayons :

- en **[1]**, le dossier **[src]** a été renommé en **[vuejs-00]** ;
- en **[2-3]**, on a modifié la commande **[serve]** ;
- en **[4-6]**, on exécute le projet ;

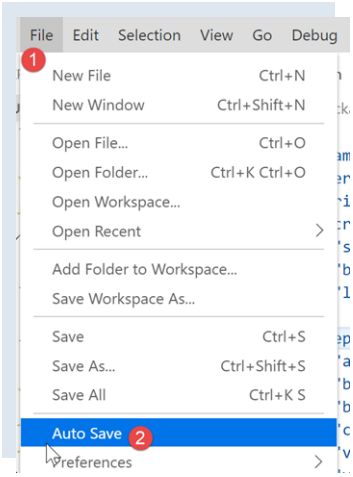
On obtient le même résultat que précédemment :



Pour nos tests, nous procéderons donc ainsi :

- écriture de code dans un dossier **[vuejs-xx]** du projet ;
- test de ce projet avec la commande **[vue-cli-service serve vuejs-xx/main.js]** dans le fichier **[package.json]** ;

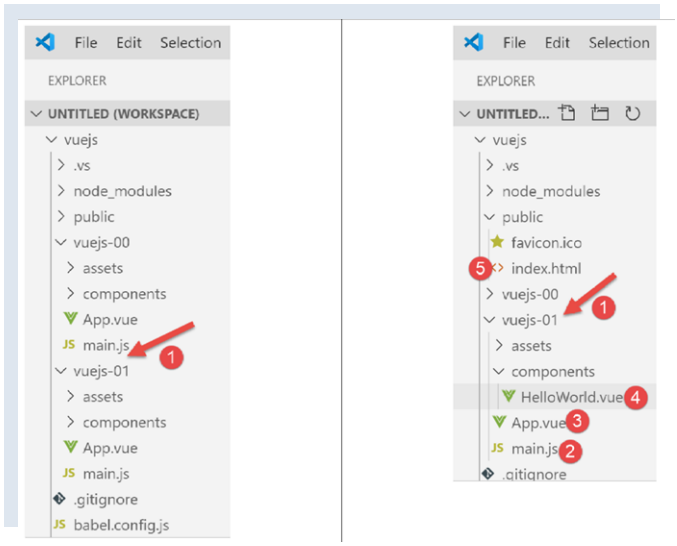
Lorsque le serveur de développement est lancé, toute modification d'un des fichiers du projet provoque une recompilation. Pour cette raison, nous inhibons le mode **[Auto Save]** de **[VSCode]**. En effet, nous ne voulons pas de recompilation dès qu'on tape des caractères dans un des fichiers du projet. Nous ne voulons de recompilation qu'à certains moments :



- en **[2]**, l'option **[Auto Save]** ne doit pas être cochée ;

3.3 projet [vuejs-01] : les bases

Pour expliquer le code exécuté dans **[vuejs-00]** nous allons le simplifier dans **[vuejs-01]**. Nous dupliquons le dossier **[vuejs-00]** dans **[vuejs-01]** :



Le projet **[vuejs-01]** comprend essentiellement quatre fichiers :

- **[main.js] [2]** est le point d'entrée du projet ;
- **[App.vue, HelloWorld.vue] [3-4]** sont des composants **[Vue.js]**, comprenant de façon facultative les éléments suivants :
  - **[<template>...</template>]** : du code HTML ;
  - **[<script>...</script>]** : le code Javascript associé au code HTML ;
  - **[<style>...</style>]** : le style CSS associé au code HTML ;
- **[public/index.html] [5]** : le document HTML visualisé par la commande **[npm run serve]** ;

Le fichier **[public/index.html]** affiché à l'exécution du projet est celui-ci :

```
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="utf-8">
5       <meta http-equiv="X-UA-Compatible" content="IE=edge">
6       <meta name="viewport" content="width=device-width,initial-scale=1.0">
7       <link rel="icon" href="<%= BASE_URL %>favicon.ico">
8       <title>vuejs</title>
9     </head>
10    <body>
11      <noscript>
12        <strong>We're sorry but vuejs doesn't work properly without JavaScript enabled. Please enable it to
    continue.</strong>
13      </noscript>
14      <div id="app"></div>
15      <!-- built files will be auto injected -->
16    </body>
17  </html>
```

Ce fichier HTML n'affiche donc rien **statiquement**. Il n'y a ici pas de code HTML. L'affichage est **dynamique** : le code jS du projet va générer du HTML qui va remplacer entièrement la balise **[ ]** de la ligne 14. Le code HTML généré par le code jS du projet et inséré à la place de la balise **[ ]** de la ligne 14 provient des balises **[template]** des composants **[vue.js]**, les fichiers ayant le suffixe **[.vue]**.

Le code HTML est inséré dynamiquement ligne 14 par le script **[vuejs-01/main.js]** suivant :

```
1   // imports
2   import Vue from 'vue'
3   import App from './App.vue'
4
5   // configuration
6   Vue.config.productionTip = false
7
8   // instanciation projet [App]
9   new Vue({
10    render: h => h(App),
11  }).$mount('#app')
```

Commentaires

- ligne 2 : l'objet **[Vue]** est fourni par le framework **[vue.js]** ;
- ligne 3 : l'objet **[App]** est fourni par le fichier **[vuejs-01/App.vue]** ;
- ligne 6 : configuration de l'objet **[Vue]** ;
- lignes 9-11 : ce sont les lignes qui :
  - génèrent le code HTML de l'application. Ligne 10, c'est le fichier **[App.vue]** qui le génère ;
  - chargent le code HTML généré ligne 10 dans la section **[ ]** du fichier **[public/index.html]** ;

Tout projet **[Vue.js]** peut conserver le fichier **[index.html]** tel quel.

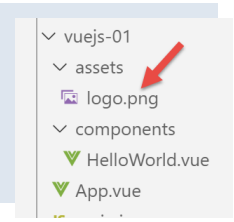
Le fichier **[App.vue]** du projet initial **[vuejs-00]** est simplifié de la façon suivante dans le projet **[vuejs-01]** :

```
1.  <template>
2.    <div id="app">
3.      <img alt="Vue logo" src="./assets/logo.png">
4.      <HelloWorld msg="Welcome to Your Vue.js App"/>
5.    </div>
6.  </template>
7.
8.  <script>
9.  import HelloWorld from './components/HelloWorld.vue'
10.
11. export default {
12.   name: 'app',
13.   components: {
14.     HelloWorld
15.   }
16. }
17. </script>
```

Commentaires

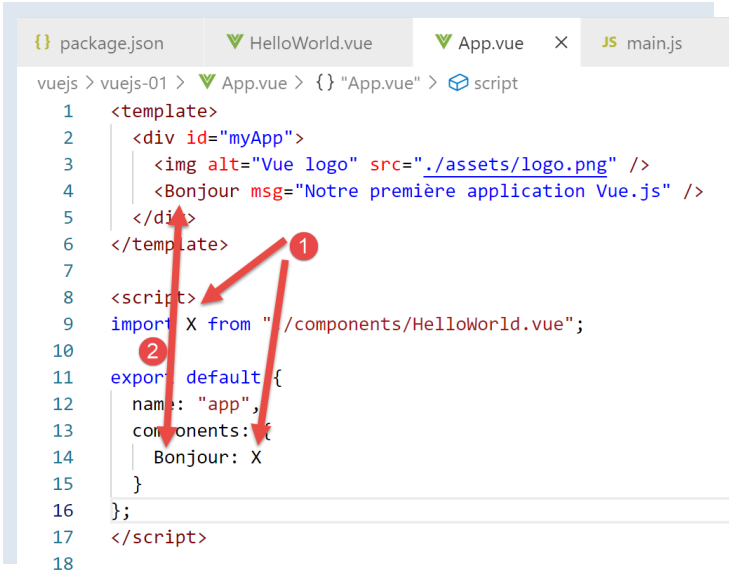
- un fragment **[.vue]** comprend au plus trois sections :
  - **[<template>...</template>]** : du code HTML ;
  - **[<script>...</script>]** : le code Javascript associé au code HTML ;
  - **[<style>...</style>]** : le style CSS associé au code HTML ;

  Ici, nous n'avons pas de section **[style]**.
- lignes 1-6 : le code HTML du fragment (page, composant, vue, ...) ;
- lignes 2-5 : la section **[template]** ne peut contenir qu'un élément. On met en général une section **[div]** qui englobe tout le HTML du fragment. On peut mettre également une balise <template> ;
- ligne 3 : une image ;



- ligne 4 : un composant nommé **[HelloWorld]**. Le principe de **[Vue.js]** est de construire des pages web à l'aide de fragments définis dans des fichiers **[.vue]** comme ici **[App.vue]**. Ce composant est défini par le fichier **[HelloWorld.vue]** défini ligne 9 du script jS associé ;
- ligne 4 : un composant peut accepter des paramètres. Le paramètre est ici l'attribut **[msg]** ;
- lignes 8-17 : le script jS du fragment (ou composant) ;
- ligne 9 : pour pouvoir utiliser le composant **[HelloWorld]** dans le composant **[App]**, il faut importer sa définition dans la partie **[script]** ;
- lignes 11-16 : le script définit un objet et l'exporte afin de le rendre disponible à l'extérieur ;
- ligne 12 : l'attribut **[name]** : définit le nom du composant exporté ;
- lignes 13-15 : l'attribut **[components]** liste les composants utilisés par le composant **[App]**. Ils sont exportés avec lui ;

Ligne 9, il n'y a pas obligation que le composant **[HelloWorld]** porte le même nom que le fichier qui le définit. On pourrait l'importer en tant que **[X]** et l'exporter en tant que composant **[Bonjour]** :

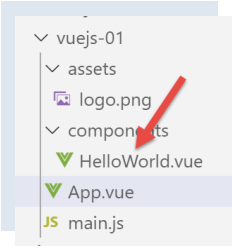


- ligne 14 : le composant **[X]** est exporté sous le nom **[Bonjour]**. Il est alors utilisé sous ce nom, ligne 4 ;

La première version est la version la plus courante, aussi définirons-nous nos composants de cette façon ;

```
1  <template>
2    <div id="myApp">
3      <img alt="Vue logo" src="./assets/logo.png" />
4      <HelloWorld msg="Notre première application Vue.js" />
5    </div>
6  </template>
7
8  <script>
9  import HelloWorld from "./components/HelloWorld.vue";
10
11 export default {
12   name: "app",
13   components: {
14     HelloWorld
15   }
16 };
17 </script>
```

La ligne 14 est un raccourci pour le code **[HelloWorld : HelloWorld]** : le composant **[HelloWorld]** (à droite, importé ligne 9) est exporté sous le nom **[HelloWorld]** (à gauche).



Nous simplifions le composant **[HelloWorld.vue]** de la façon suivante :

```
1  <template>
2    <div>
3      <h1>{{ msg }}</h1>
4    </div>
5  </template>
6
7  <script>
8  export default {
9    name: "HelloWorld",
```

```
10    props: {
11      msg: String
12    }
13  };
14  </script>
```

Commentaires

- le composant **[HelloWorld]** a la même structure de fichier que le composant principal **[App]** ;
- ligne 3 : on a ici une évaluation d'expression Javascript, ici l'expression **[msg]** ;
- lignes 10-12 : définissent les propriétés du composant, plus exactement ses paramètres. Lorsque le composant **[App]** a instancié un composant **[HelloWorld]**, il l'a fait avec la syntaxe suivante :

```
<HelloWorld msg="Notre première application Vue.js" />
```

  Le composant **[HelloWorld]** est instancié en donnant une valeur au paramètre (attribut) **[msg]**. Si on suit le **[template]** du composant **[HelloWorld]**, celui-ci devient :

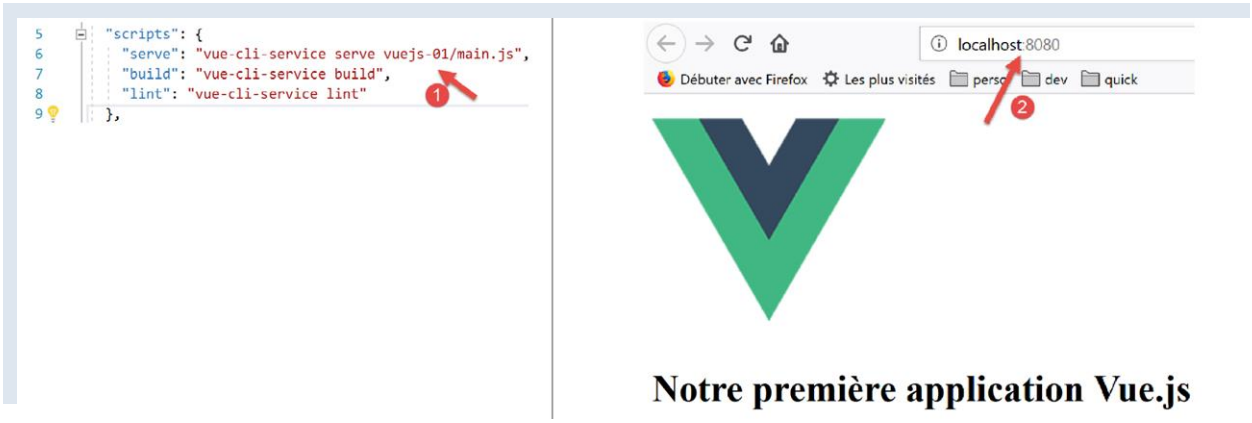
```
<div>
    <h1>Notre première application Vue.js</h1>
</div>
```

- lignes 7-14 : les propriétés du composant définies sous la forme d'un objet qui est exporté ;
  - ligne 9 : le composant est exporté sous le nom **[HelloWorld]** ;
  - lignes 10-12 : ses paramètres sont définis par la propriété **[props]** ;

Au final, si on rassemble les templates des deux composants **[App, HelloWorld]** utilisés, le fichier **[index.html]** affiché sera le suivant :

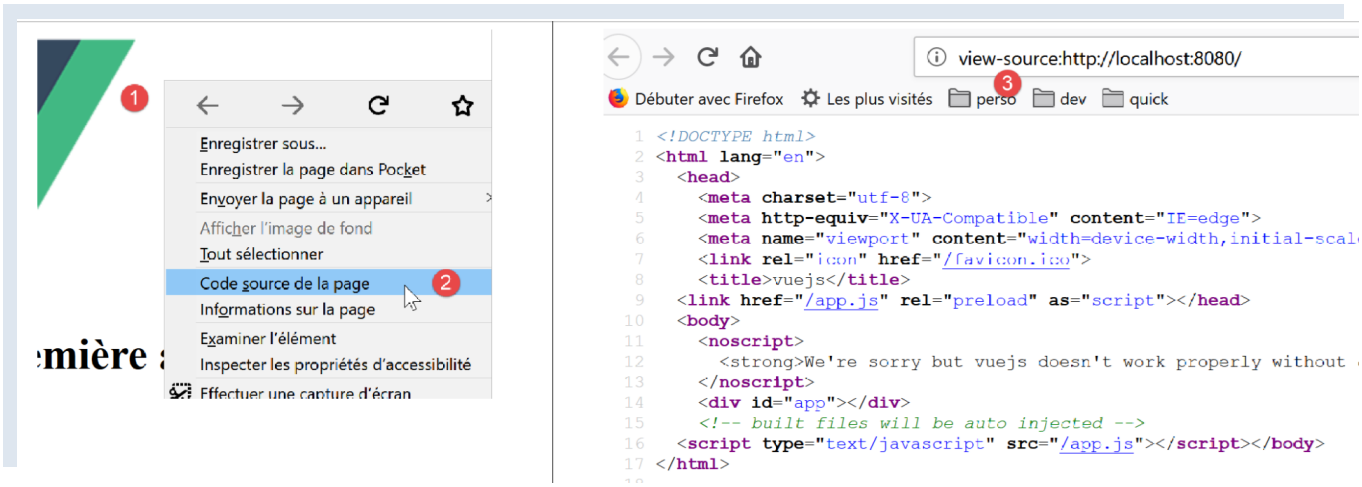
```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width,initial-scale=1.0">
7     <link rel="icon" href="<%= BASE_URL %>favicon.ico">
8     <title>vuejs</title>
9   </head>
10  <body>
11    <noscript>
12      <strong>We're sorry but vuejs doesn't work properly without JavaScript enabled. Please enable it to
    continue.</strong>
13    </noscript>
14    <div id="myApp">
15      <img alt="Vue logo" src="./assets/logo.png" />
16      <div>
17        <h1>Notre première application Vue.js</h1>
18      </div>
19    </div></body>
20  </html>
```

Nous lançons l'application en modifiant la commande **[serve] [1]** du fichier **[package.json]** :



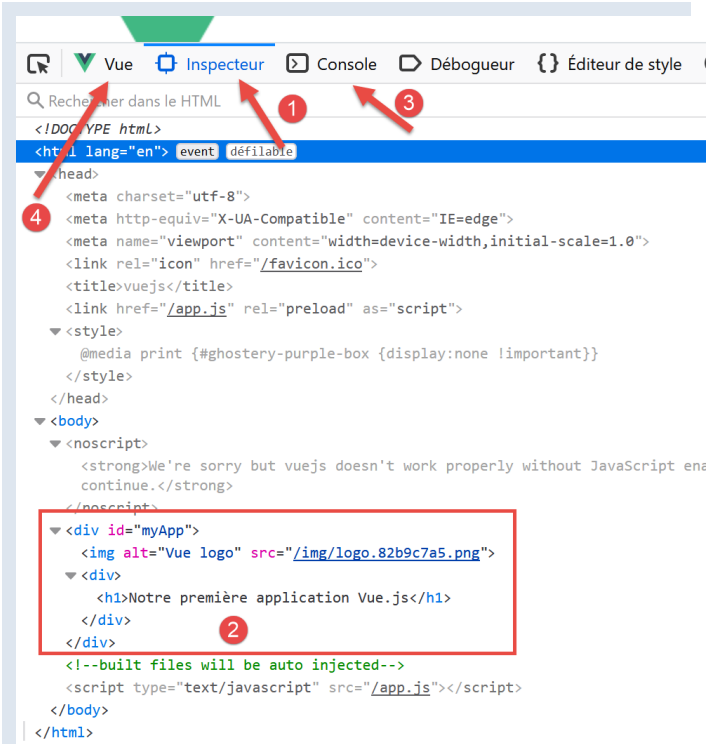
La page affichée est alors **[2]**.

Maintenant regardons le code de cette page :



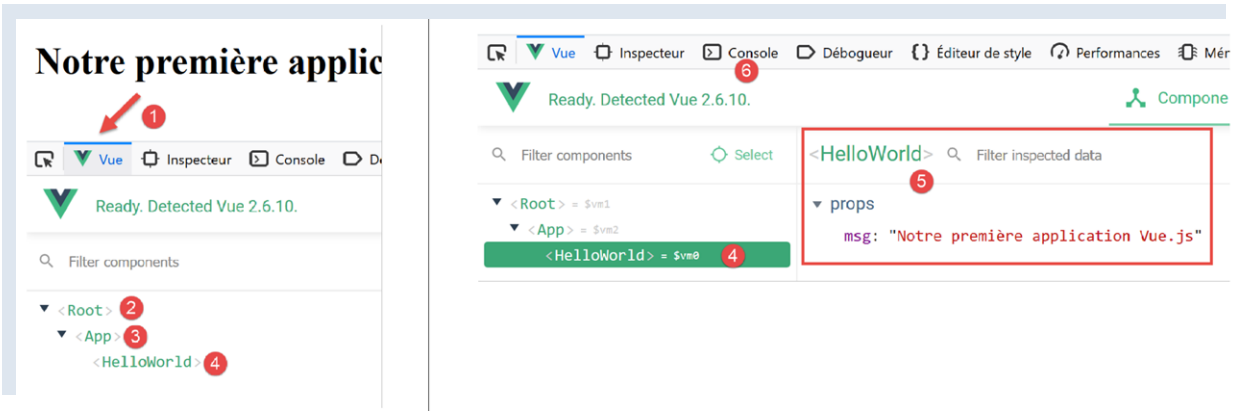
- en **[1]**, faire **[clic droit]** ;
- en **[2]**, le code source de la page. On voit que c'est le code du fichier initial **[index.html]** et ce n'est pas ça qui a été affiché. C'est bien la page **[index.html]** qui a été chargée initialement. Ensuite, dynamiquement, du code Javascript a modifié cette page, mais cela ne nous est pas montré ;

Lorsque les pages sont générées dynamiquement par du Javascript, l'option **[2]** ne sert à rien. Il faut aller dans les outils du navigateur (F12 sur Firefox) pour voir le code de la page actuellement affichée :



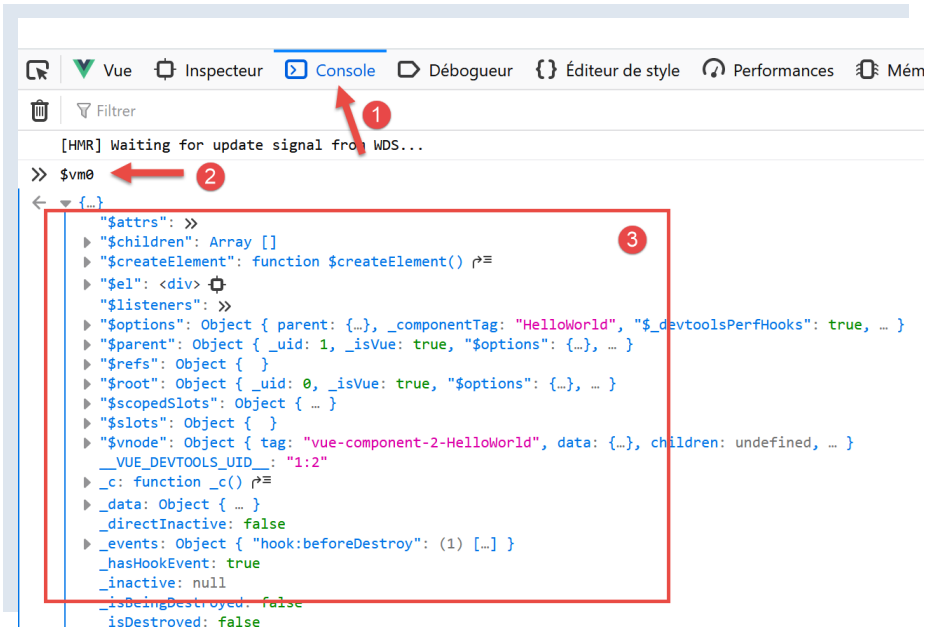
- en **[1]**, l'inspecteur du DOM (Document Object Model) du document affiché ;
- en **[2]**, ce que contient réellement ce DOM ;
- **[3-4]**, des outils que nous utiliserons pour afficher les objets Javascript utilisés par le framework **[Vue.js]** ;
- **[4]** est une extension (ici Firefox) pour déboguer des applications **[Vue.js]** :
  - pour Firefox : **[https://addons.mozilla.org/fr/firefox/addon/vue-js-devtools/]**;
  - pour Chrome : **[https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd]**;

Examinons l'onglet **[Vue] [4]** :



La vue **[1-4]** nous montre la structure **[Vue.js]** du document : la racine du document **[2]** (index.html) comprend le composant **[App]** (3) qui lui même comprend le composant **[HelloWorld]** (4). Cliquer sur **[4]** fait apparaître les propriétés du composant **[HelloWorld]** **[5]**.

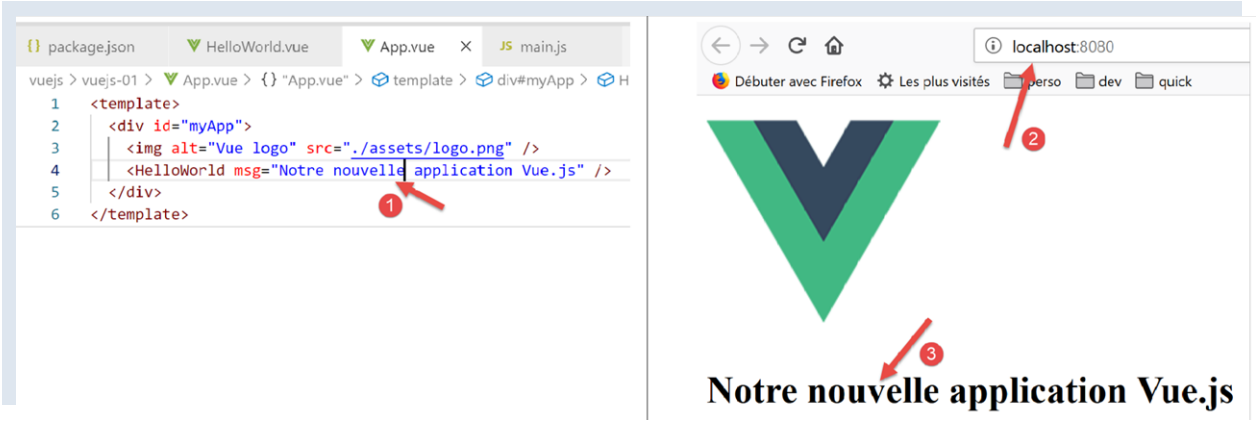
On voit en **[4]** (à droite), l'indicateur **[$vm0]**. C'est le nom de la variable qu'on peut utiliser dans la console JavaScript **[6]** pour désigner l'objet **[HelloWorld]**. Faisons-le :



- en **[2]**, on fait évaluer l'expression **[$vm0]**, ce qui a pour effet d'afficher sa structure. Normalement nous n'aurons pas à utiliser directement cette structure ;

Terminons en montrant la capacité de **[hot reload]** de la commande **[serve]** utilisée pour exécuter le projet :

- dans **[App.vue]**, modifiez le message affiché par **[HelloWorld]** :



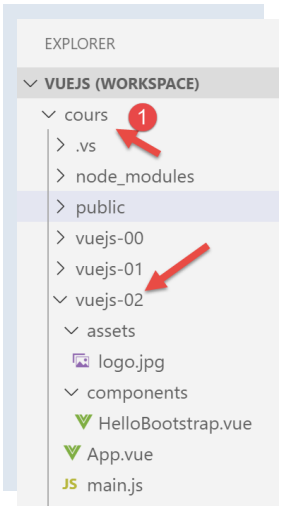
- en **[1]**, on modifie le message affiché ;
- en **[2-3]**, la page est automatiquement mise en jour sans intervention de notre part ;

Nous allons maintenant créer divers projets **[vuejs-xx]** pour illustrer les points importants de **[Vue.js]**. Par 'importants', il faut entendre 'que nous allons utiliser dans le client **[vue.js]** du serveur de calcul de l'impôt'. D'autres points 'importants' seront passés sous silence s'ils ne sont pas utilisés dans le client. Ce n'est donc pas une présentation exhaustive de **[vue.js]** qui sera faite.

3.4 projet [vuejs-02] : utilisation du framework CSS Bootstrap

Le projet **[vuejs-02]** présente l'utilisation de Bootstrap dans un projet **[vue.js]**. C'est le framework CSS qui sera utilisé dans tous nos projets. Nous utiliserons une variation de Bootstrap appelée **[BootstrapVue]** **[https://bootstrap-vue.js.org/]**.

L'arborescence du projet sera la suivante :



Note : ci-dessus le dossier **[vuejs]** a été renommée **[cours]** **[1]** dans la suite du document.

3.4.1 Installation du framework [BootstrapVue]

[BootstrapVue] est un framework qu'on ajoute au projet avec l'outil **[npm]** :

- en **[1]**, c'est donc deux frameworks qu'on installe : **[Bootstrap]** et sa variante **[BootstrapVue]** ;
- en **[2]**, les deux dépendances apparaissent dans le fichier **[package.json]** ;

3.4.2 Le script [main.js]

Le script principal **[main.js]** est le suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // plugins
6.  import BootstrapVue from 'bootstrap-vue'
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import 'bootstrap/dist/css/bootstrap.css'
11. import 'bootstrap-vue/dist/bootstrap-vue.css'
12.
13. // configuration
14. Vue.config.productionTip = false
15.
16. // instanciation projet [App]
17. new Vue({
18.   render: h => h(App),
19. }).$mount('#app')
```

- ligne 2 : import du framework **[Vue]** ;
- ligne 3 : import de la vue principale ;
- ligne 6 : import du framework **[BootstrapVue]** ;
- ligne 7 : ce framework est conçu comme un plugin du framework **[Vue]**. La ligne 7 inclut ce plugin dans le framework **[Vue]** ;
- lignes 10-11 : import des fichiers CSS des frameworks **[Bootstrap]** et **[BootstrapVue]** ;
- les lignes 5-11 sont donc entièrement consacrées à l'utilisation de **[BootstrapVue]**. Le reste du code est identique à ce qu'on avait vu au paragraphe précédent ;

3.4.3 Le composant [App.vue]

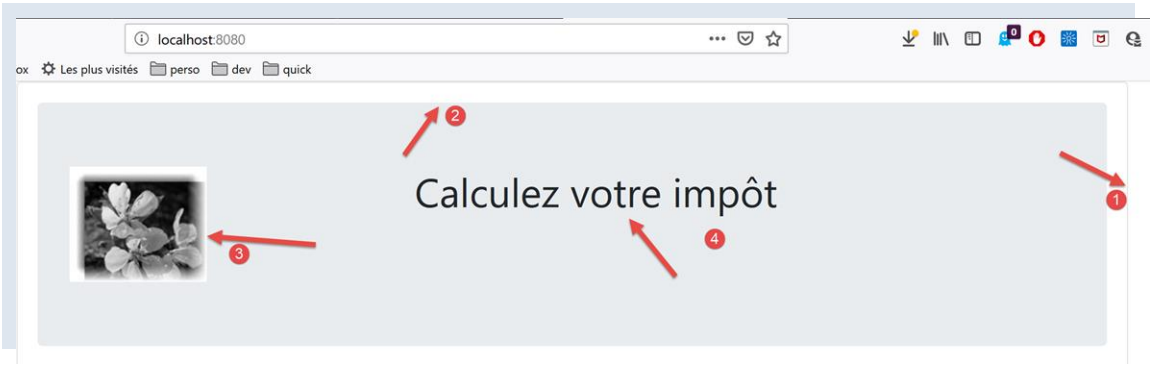
```
1.  <template>
2.    <b-container>
3.      <b-card>
4.      <!-- Bootstrap Jumbotron -->
5.      <b-jumbotron>
6.        <!-- ligne -->
7.        <b-row>
8.          <!-- colonne de largeur 4 -->
9.          <b-col cols="4">
10.           <img src="./assets/logo.jpg" alt="Cerisier en fleurs" />
11.         </b-col>
12.         <!-- colonne de largeur 8 -->
13.         <b-col cols="8">
14.           <h1>Calculez votre impôt</h1>
15.         </b-col>
16.       </b-row>
```

```
17.     </b-jumbotron>
18.     <HelloBootstrap msg="Hello Bootstrap !" />
19.     </b-card>
20.   </b-container>
21. </template>
22.
23. <script>
24. import HelloBootstrap from "./components/HelloBootstrap.vue";
25.
26. export default {
27.   name: "app",
28.   components: {
29.     HelloBootstrap
30.   }
31. };
32. </script>
```

Commentaires

- lignes 1-21 : toutes les balises <b-xx> sont des balises du framework **[BootstrapVue]** ;
- lignes 2, 20 : la balise <b-container> définit un conteneur Bootstrap. A l'intérieur de ce conteneur, on va pouvoir définir des lignes avec la balise <b-row> et des colonnes avec la balise <b-col> ;
- lignes 3, 19 : la balise <b-card> définit une 'carte' Bootstrap. Cela se matérialise visuellement par un rectangle avec une bordure ;
- lignes 5, 17 : la balise <b-jumbotron> permet de mettre en avant une partie de la page, ici une image et un texte. On l'utilisera dans nos divers projets comme identification visuelle du projet ;
- ligne 7 : la balise <b-row> définit une ligne ;
- lignes 9-11 : la balise <b-col> définit une colonne de la ligne précédente. Bootstrap attribue 12 colonnes à chaque ligne. L'attribut **[cols='4']** indique que la colonne <b-col> va occuper 4 de ces 12 colonnes ;
- ligne 10 : une image
- lignes 13-15 : une colonne qui va occuper 8 des 12 colonnes de la ligne. on y met un texte ;
- ligne 18 : utilisation d'un composant appelé **[HelloBootstrap]** avec une propriété nommée **[msg]** ;
- lignes 23-31 : la partie <script> du composant ;
- lignes 29-31 : le composant **[HelloBootstrap]** utilisée ligne 18 est exporté. Pour être connu, il doit être importé ligne 245 ;

Le résultat est le suivant :



- en **[1]**, la balise <b-card> ;
- en **[2]**, la balise <jumbotron> ;
- en **[3]**, l'image sur 4 colonnes ;
- en **[4]**, le texte sur 8 colonnes ;

3.4.4 Le composant [HelloBootstrap]

[HelloBootstrap] est le composant suivant :

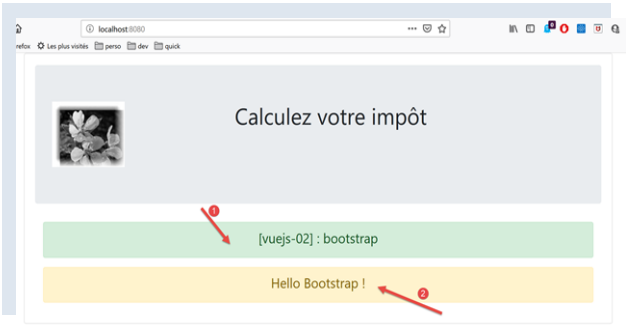
```
1.  <template>
2.    <div>
3.      <!-- message sur fond vert -->
4.      <b-col cols="12">
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-02] : bootstrap</h4>
7.        </b-alert>
8.      </b-col>
9.      <!-- message sur fond jaune -->
10.     <b-col cols="12">
```

```
11.       <b-alert show variant="warning" align="center">
12.         <h4>{{msg}}</h4>
13.       </b-alert>
14.     </b-col>
15.   </div>
16. </template>
17.
18. <script>
19.   export default {
20.     name: "HelloBootstrap",
21.     props: {
22.       msg: String
23.     }
24.   };
25. </script>
```

Commentaires

- ligne 3 : la balise <b-alert show> affiche un rectangle de couleur dans lequel on met en général un texte (ligne 6). L'attribut **[variant]** permet de sélectionner un type d'alerte. Chaque type d'alerte a une couleur de fond différente. La couleur de la variante **[success]** est le vert. L'attribut **[align]** permet d'aligner le texte de l'alerte (gauche, droite, centré). On notera que l'attribut **[show]** est obligatoire pour afficher l'alerte. Sans cet attribut, l'alerte n'est pas visible ;
- les valeurs possibles de **[variant]** :
  - **[primary]** : bleu ;
  - **[secondary]** : gris ;
  - **[success]**:vert ;
  - **[danger]** : rouge léger ;
  - **[warning]** : jaune ;
  - **[info]** : turquoise ;
  - **[light]** : pas de couleur de fond ;
  - **[dark]** : gris un peu plus foncé que **[secondary]** ;
- ligne 12 : **[msg]** est un paramètre du composant **[HelloBootstrap]** (lignes 21-23) ;

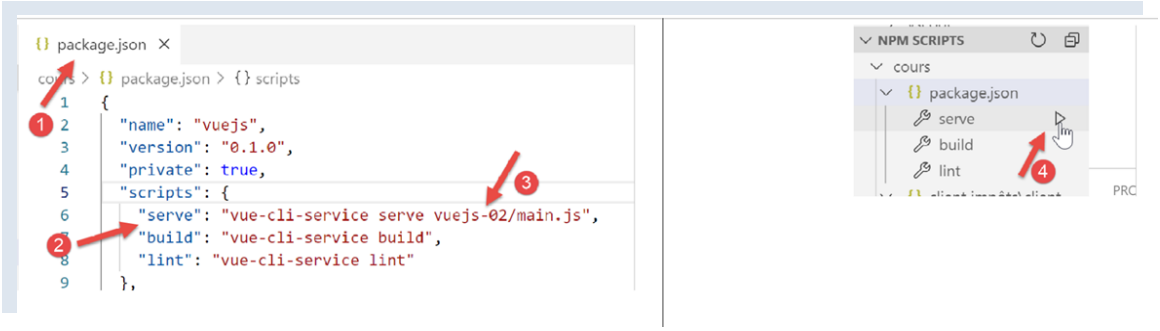
Le rendu visuel est le suivant :



- **[1]** : balise <b-alert show variant='success'> ;
- **[2]** : balise <b-alert show variant='warning'> ;

3.4.5 Exécution du projet

Pour exécuter le projet, on modifie d'abord le fichier **[package.json]** :



- en **[3]**, on modifie le script exécuté par la commande **[serve]** **[2]** du fichier package.json **[1]** ;
- en **[4]**, on exécute le projet ;

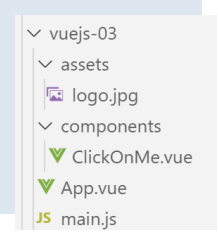
Note : dans tout ce qui suit on utilisera les balises du framework BootstrapVue, des balises de la forme <b-qqchose>. Ce n'est pas obligatoire. On peut utiliser les balises originelles du framework Bootstrap. Elles sont fonctionnelles dans les templates de **[Vue.js]**. Aussi le développeur habitué aux balises Bootstrap peut continuer à les utiliser.

3.5 projet [vuejs-03] : gestion des événements

Le projet **[vuejs-03]** introduit deux concepts :

- la gestion d'un événement **[clic]** sur un bouton ;
- la directive **[v-if]** qui permet d'afficher un bloc HTML de façon conditionnelle ;

L'arborescence du projet est la suivante :



3.5.1 Le script principal [main.js]

Le script **[main.js]** reste inchangé :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // plugins
6.  import BootstrapVue from 'bootstrap-vue'
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import 'bootstrap/dist/css/bootstrap.css'
11. import 'bootstrap-vue/dist/bootstrap-vue.css'
12.
13. // configuration
14. Vue.config.productionTip = false
15.
16. // instanciation projet [App]
17. new Vue({
18.   render: h => h(App),
19. }).$mount('#app')
```

3.5.2 Le composant principal [App.vue]

Le composant principal **[App.vue]** utilise le composant **[ClickOnMe]** au lieu du composant **[HelloBootstrap]** :

```
1.  <template>
2.    <b-container>
3.      <b-card>
4.      <!-- Bootstrap Jumbotron -->
5.      <b-jumbotron>
6.        <!-- ligne -->
7.        <b-row>
8.          <!-- colonne de largeur 4 -->
9.          <b-col cols="4">
10.           <img src="./assets/logo.jpg" alt="Cerisier en fleurs" />
11.         </b-col>
12.         <!-- colonne de largeur 8 -->
13.         <b-col cols="8">
14.           <h1>Calculez votre impôt</h1>
15.         </b-col>
16.       </b-row>
```

```
    </b-jumbotron>
    <!-- composant -->
    <ClickOnMe msg="Information..." />
    </b-card>
  </b-container>
</template>

<script>
import ClickOnMe from "./components/ClickOnMe.vue";

export default {
  name: "app",
  components: {
    ClickOnMe
  }
};
</script>
```

3.5.3 Le composant `[ClickOnMe]`

Le composant **`[ClickOnMe]`** introduit les nouveaux concepts :

```
<template>
  <div>
    <!-- message sur fond vert -->
    <b-alert show variant="success" align="center">
      <h4>[vuejs-03] : événement @click, directive v-if, méthodes</h4>
    </b-alert>
    <!-- message sur fond jaune -->
    <b-alert show variant="warning" align="center" v-if="show">
      <h4>{{msg}}</h4>
    </b-alert>
    <!-- bouton bleu -->
    <b-button variant="primary" @click="changer">{{buttonTitle}}</b-button>
  </div>
</template>

<script>
  export default {
    name: "ClickOnMe",
    // paramètres du composant
    props: {
      msg: String
    },
    // attributs du composant
    data() {
      return {
        // titre du bouton
        buttonTitle: "Cacher",
        // contrôle l'affichage du message
        show: true
      };
    },
    // méthodes
    methods: {
      // montre / cache le message
      changer() {
        if (this.show) {
          // on cache le message
          this.show = false;
          this.buttonTitle = "Montrer";
        } else {
          // on montre le message
          this.show = true;
          this.buttonTitle = "cacher";
        }
      }
    }
  };
</script>
```

Commentaires

- lignes 4-6 : une alerte verte Bootstrap. Le nombre de colonnes occupées n'est pas indiqué. Ce sont alors les 12 colonnes de Bootstrap qui sont utilisées ;
- lignes 8-10 : une alerte jaune Bootstrap :
  - ligne 8 : la directive **[v-if]** de **[Vue.js]** contrôle la visibilité d'un bloc HTML. L'alerte est ici contrôlée par un booléen **[show]** (ligne 29). Si **[show==true]** alors l'alerte sera visible sinon elle ne le sera pas ;
  - ligne 9 : l'alerte affiche un message **[msg]** qui est une propriété (lignes 20-22) du composant ;
- ligne 12 : un bouton de couleur bleue sur laquelle on clique pour cacher / montrer l'alerte **[warning]** ;
- lignes 16-48 : le code jS du composant. Ce code règle le fonctionnement dynamique du composant :
- lignes 20-22 : les propriétés du composant ;
- lignes 24-31 : les attributs du composant ;

Quelle est la différence entre **[propriétés]** et **[attributs]** d'un composant, entre les champs **[props]** et **[data]** de l'objet exporté par le composant aux lignes 17-47 ?

- comme nous l'avons déjà vu, les propriétés **[props]** d'un composant sont des paramètres du composant. Leurs valeurs sont fixées de l'**extérieur** du composant. Un composant A utilisant un composant B ayant les propriétés **[prop1, prop2, ..., propn]** l'utilisera de la façon suivante : **<B :prop1='val1' :prop2='val2' ...>** ;
- l'objet rendu par la fonction **[data]** des lignes 24-31 représente l'état du composant ou **attributs** du composant. Cet état est manipulé par les méthodes du composant (lignes 33-46). Le <template> des lignes 1-14 utilise aussi bien des éléments **[propriétés]** que **[attributs]** :
  - les valeurs des propriétés sont fixées par un composant parent ;
  - les valeurs des attributs sont fixées initialement par la fonction **[data]** puis peuvent être modifiées par les méthodes ;
  - dans les deux cas, le rendu visuel **réagit** immédiatement aux changements d'une propriété (composant parent) ou d'un attribut (méthode du composant). On parle alors d'interface **réactive** ;

  Dans le **[template]** d'un composant, rien ne diffère une propriété **[prop]** d'un attribut **[data]**. Pour savoir si une donnée dynamique du **[template]** doit être mise dans l'attribut **[props]** ou dans l'objet rendu par la fonction **[data]**, il faut simplement se demander qui fixe la valeur de cette donnée :

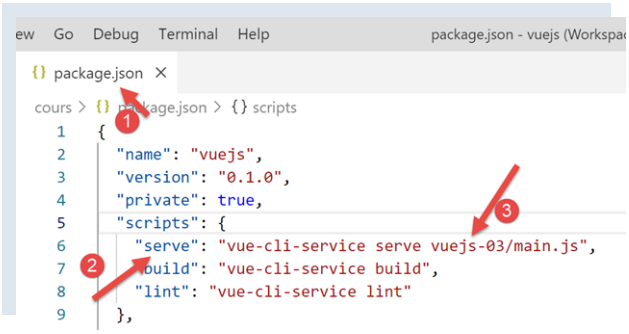
  - si la réponse est le composant parent, alors on mettra la donnée dans l'attribut **[props]** ;
  - si la réponse est la méthode gérant tel événement du composant, alors on mettra la donnée dans l'objet rendu par la fonction **[data]** ;

  Le **[template]** utilise ici les données dynamiques suivantes :

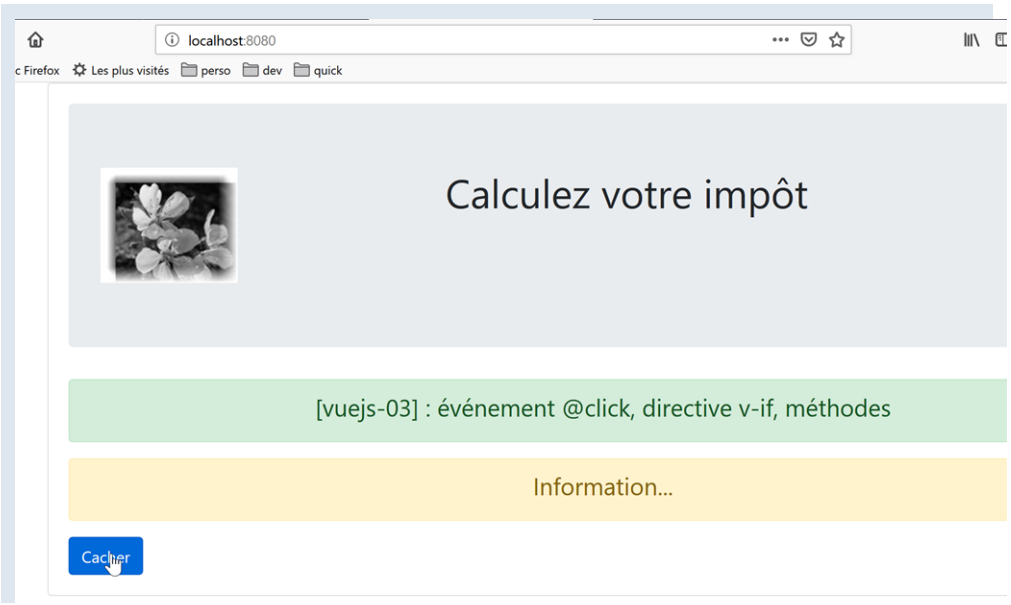
  - **[show]**, ligne 8. Cette donnée est manipulée en interne par la méthode **[changer]** qui gère l'événement **[click]** sur le bouton de la ligne 12. C'est donc un attribut construit par la fonction **[data]** (ligne 29) ;
  - **[msg]**, ligne 9. C'est un message fixé par le composant parent. On le met donc dans l'attribut **[props]** (ligne 21) ;
  - **[buttonTitle]** ligne 12. Cette donnée est manipulée en interne par la méthode **[changer]** qui gère l'événement **[click]** sur le bouton de la ligne 12. C'est donc un attribut construit par la fonction **[data]** (ligne 27) ;

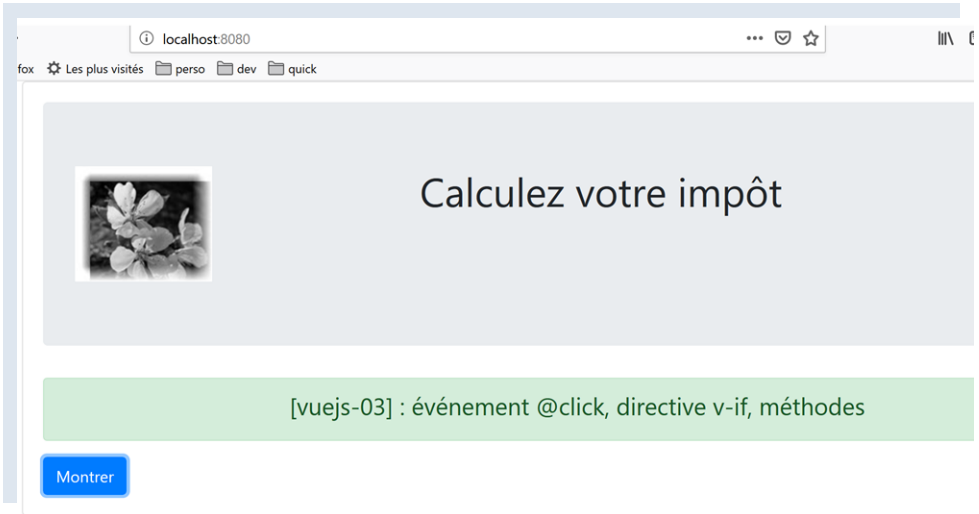
- les noms des attributs **[name, props, data, methods]** de l'objet exporté par le composant sont prédéfinis. On ne peut pas utiliser d'autres noms ;
- ligne 12 : l'attribut **[@click]** du bouton sert à désigner la méthode qui doit réagir au clic sur le bouton. Cette méthode doit se trouver dans la propriété **[methods]** du composant ;
- ligne 33 : l'attribut **[methods]** du composant réunit toutes les méthodes de celui-ci. La plupart du temps ce sont des fonctions qui réagissent à un événement du composant ;
- lignes 35-46 : la méthode **[changer]** est appelée lorsque l'utilisateur clique sur le bouton :
  - si l'alerte **[warning]** est affichée alors elle est cachée et le texte du bouton devient **[Montrer]** (ligne 39) ;
  - si l'alerte **[warning]** est cachée alors elle est affichée et le texte du bouton devient **[Cacher]** (ligne 43) ;
  - pour afficher / cacher l'alerte **[warning]**, on modifie la valeur du booléen **[show]** (lignes 38 et 42) ;
  - lorsqu'une méthode doit référencer l'attribut **[attr]** rendu par la fonction **[data]**, on écrit **[this.attr]** (lignes 38 et 42). Cela signifie que les attributs de l'objet rendu par la fonction **[data]** sont des attributs directs du composant **[this]** ;

3.5.4 Exécution du projet



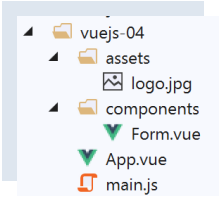
Le résultat est le suivant :





3.6 projet `[vuejs-04]` : directives `[v-model, v-bind]`, attributs calculés, formulaire de saisie

L'arborescence du projet **`[vuejs-04]`** est la suivante :



3.6.1 Le script principal [main.js]

C'est le même que dans l'exemple précédent.

3.6.2 Le composant principal [App]

Le code de **[App.vue]** est le suivant :

```
1.  <template>
2.    <b-container>
3.      <b-card>
4.        <!-- message de présentation -->
5.        <b-row>
6.          <b-col cols="8">
7.            <b-alert show variant="success" align="center">
8.              <h4>[vuejs-04] : directives [v-model, v-bind], attributs calculés, formulaire de saisie</h4>
9.            </b-alert>
10.         </b-col>
11.       </b-row>
12.       <Form />
13.     </b-card>
14.   </b-container>
15. </template>
16.
17. <script>
18.   import Form from "./components/Form.vue";
19.
20.   export default {
21.     name: "app",
22.     components: {
23.       Form
24.     }
25.   };
26. </script>
```

Le composant **[App.vue]** utilise le nouveau composant **[Form]** (lignes 12, 18, 23).

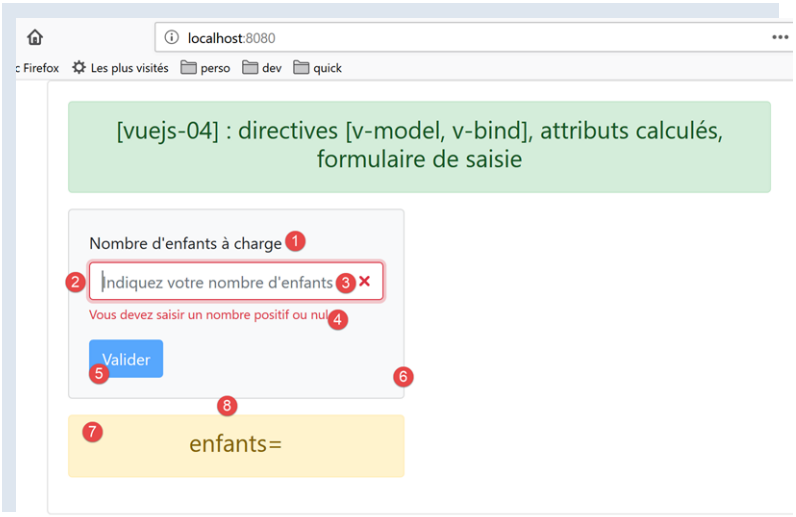
3.6.3 Le composant [Form]

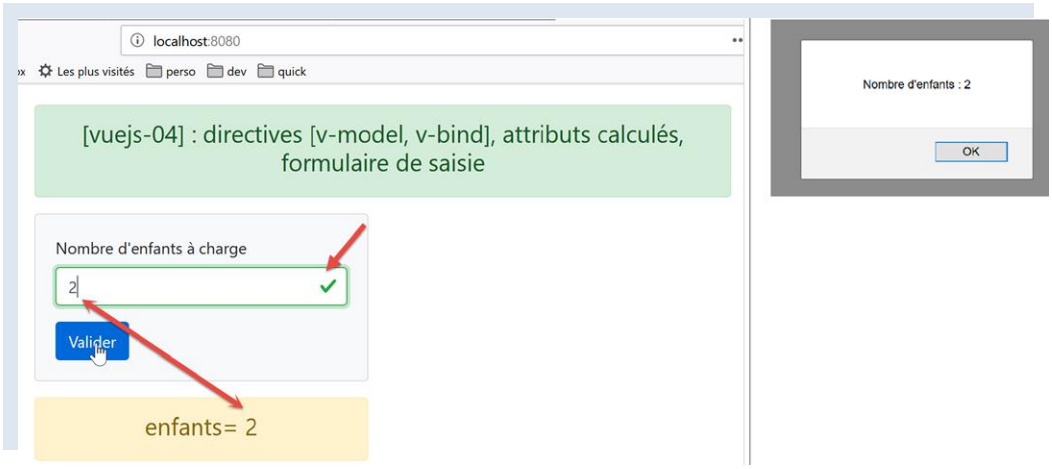
Le code du composant **[Form]** est le suivant :

```
1.  <template>
2.    <div>
3.      <!-- formulaire -->
4.      <b-form>
5.        <!-- éléments du formulaire -->
6.        <b-row>
7.          <b-col cols="4">
8.            <b-card bg-variant="light">
9.              <b-form-group label="Nombre d'enfants à charge" label-for="enfants">
10.               <b-input type="text" required
11.                        id="enfants"
12.                        placeholder="Indiquez votre nombre d'enfants"
13.                        v-model="enfants"
14.                        v-bind:state="enfantsValide" />
15.             <b-form-invalid-
    feedback :state="enfantsValide">Vous devez saisir un nombre positif ou nul</b-form-invalid-feedback>
16.             </b-form-group>
17.             <!-- bouton [submit] -->
18.             <b-button variant="primary" :disabled="formInvalide" @click="doSomething">Valider</b-button>
19.           </b-card>
20.         </b-col>
21.       </b-row>
22.     </b-form>
```

```
23.     <b-row class="mt-3">
24.       <b-col cols="4">
25.         <b-alert show variant="warning" align="center">
26.           <h4>enfants= {{enfants}}</h4>
27.         </b-alert>
28.       </b-col>
29.     </b-row>
30.   </div>
31. </template>
32.
33. <!-- script -->
34. <script>
35.   export default {
36.     // nom
37.     name: "Form",
38.     // attributs statiques du composant
39.     data() {
40.       return {
41.         // nbre d'enfants
42.         enfants: ""
43.       };
44.     },
45.
46.     // attributs calculés
47.     computed: {
48.       // attribut [formInvalide]
49.       formInvalide() {
50.         return !this.enfantsValide;
51.       },
52.       // attribut [enfantsInvalide]
53.       enfantsValide() {
54.         return Boolean(this.enfants.match(/^\s*\d+\s*$/));
55.       }
56.     },
57.     // méthodes
58.     methods: {
59.       doSomething() {
60.         // le nbre d'enfants est connu lorsque la validation a lieu
61.         alert("Nombre d'enfants : " + this.enfants);
62.       }
63.     }
64.   };
65. </script>
```

Rendu visuel





Commentaires

- lignes 4-32 : la balise <b-form> introduit un formulaire Bootstrap ;
- ligne 6 : la balise <b-row> introduit une ligne dans le formulaire ;
- ligne 7 : la balise <b-col cols='4'> introduit une colonne s'étalant sur 4 colonnes Bootstrap ;
- ligne 8 : la balise <b-card> **[6]** introduit une carte Bootstrap, une zone encadrée par une bordure ;
- ligne 9 : la balise <b-form-group> introduit un groupe d'éléments du formulaire liés entre-eux. Ici un texte (attribut **[label]**) **[1]** est lié à une zone de saisie (attribut **[label-for]**). La valeur de **[label-for]** est la valeur du champ **[id]**, ligne 12, de la zone de saisie ;
- lignes 10-14 : la balise <b-input> **[2]** introduit une zone de saisie :
  - ligne 10 : **[type='text']** indique qu'on peut taper du texte dans la zone de saisie. On aurait pu écrire **[type='number']** avec des contraintes **[min='val1' max='val2' step='val3']** puisqu'on attend un nombre d'enfants. On a mis **[type='text']** afin de montrer comment vérifier la validité d'une saisie ;
  - ligne 12 : l'attribut **[placeholder] [3]** fixe le message affiché dans la zone de saisie tant que l'utilisateur n'a rien saisi ;
  - ligne 13 : la directive **[v-model]** associe, de façon **bidirectionnelle** la valeur saisie avec l'attribut **[enfants]**, ligne 42, du composant :
    - lorsque la valeur saisie change, alors la valeur de l'attribut **[enfants]** change également ;
    - lorsque la valeur de l'attribut **[enfants]** change, alors la valeur saisie change également, ç-à-d que le contenu de **[2]** change ;
    - le point important à comprendre est que, grâce au mécanisme précédent, lorsque l'utilisateur clique sur le bouton **[Valider] [5]**, l'attribut **[enfants]** de la ligne 42 a pour valeur, la valeur saisie en **[2]** ;
  - ligne 14 : la directive **[v-bind]** introduit une liaison entre d'un côté un attribut de la balise <b-input>, ici l'attribut **[state]** avec un attribut du composant, ici **[enfantsValide]**, ligne 53. L'attribut **[enfantsValide]** est particulier en ce sens que c'est une **fonction** qui rend la valeur de l'attribut. On appelle, attribut **calculé**, ce type d'attribut. On trouve les attributs calculés dans la propriété **[computed]**, ligne 47, du composant. Les attributs calculés s'utilisent de la même façon que les attributs statiques de la fonction **[data]** : On n'écrit pas, ligne 14, **[v-bind:state='enfantsValide()']** mais **[v-bind:state='enfantsValide']**, **sans les parenthèses**. Aussi à la lecture du **[template]**, on ne sait pas distinguer un attribut **calculé** d'un attribut **statique**. Il faut pour cela regarder le code du script du composant ;
  - ligne 14 : l'attribut **[state]** va fixer l'état valide / invalide de la valeur saisie : si **[enfantsValide]** rend la valeur **[true]**, la valeur saisie est considérée comme valide, sinon comme invalide. La copie d'écran ci-dessus montre le composant **[b-input]** lorsque la fonction **[enfantsValide]** rend la valeur **[false]** ;
  - ligne 15 : la balise <b-form-invalid-feedback> **[4]** permet d'afficher un message lorsque la saisie en **[2]** est invalide. Son attribut **[:state='enfantsValide']** est identique à l'attribut **[v-bind:state='enfantsValide']** de la ligne 14. On peut omettre la directive **[v-bind]** mais il faut garder le signe **[:]**. Le message d'erreur s'affiche donc lorsque l'attribut **[enfantsValide]** vaut **[false]** ;
  - ligne 16 : fin du groupe d'éléments <b-group> ;
  - ligne 18 : le bouton **[5]** qui va permettre de valider la saisie :
    - il sera bleu **[variant='primary']** ;
    - **[:disabled="formInvalide"]** : l'attribut **[disabled]** permet de valider / invalider le bouton. Cet attribut est lié (v-bind) à l'attribut calculé **[formInvalide]** de la ligne 49 ;
    - **[@click="doSomething"]** : lorsque l'utilisateur cliquera sur le bouton, la méthode **[doSomething]**, ligne 59, sera exécutée ;
  - lignes 19-22 : fermeture des différentes balises ouvertes ;
  - lignes 23-29 : une nouvelle ligne dans le **[template]**. **[class='mt-3']** signifie **[margin (m) top (t) égale à 3 spacers]**. **[spacer]** est une mesure d'espacement de Bootstrap. Cette classe génère l'espacement **[8]** dans la copie d'écran ci-dessus. Sans cette classe, la zone **[7]** est collée à la zone **[1-6]** ;
  - ligne 24 : une colonne occupant 4 colonnes Bootstrap ;

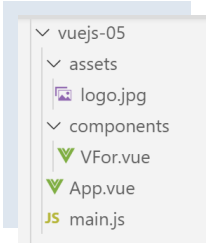
- lignes 25-27 : une alerte **[warning]** affichant la valeur de l'attribut statique **[enfants]** (ligne 42). Comme cet attribut a une liaison bidirectionnelle avec la zone de saisie, dès que l'utilisateur modifie celle-ci, la valeur affichée dans l'alerte change également ;
- lignes 34-65 : le code jS du composant ;
- ligne 42 : l'unique attribut statique du composant ;
- lignes 47-56 : les attributs calculés du composant ;
- lignes 53-55 : la saisie est considérée valide si c'est un nombre entier positif éventuellement précédé / suivi par des 'espaces' ;
- lignes 49-51 : le formulaire est considéré comme valide si la saisie du nombre d'enfants est valide. En général, un formulaire a plusieurs saisies et est considéré valide si celles-ci sont toutes valides ;
- lignes 58-63 : les méthodes du composant qui réagissent aux événements de celui-ci. Ici, il n'y a qu'un événement : le **[click]** sur le bouton. On se contente d'afficher la valeur saisie pour montrer qu'on a bien accès à celle-ci ;

3.6.4 Exécution du projet

On modifie le fichier **[package.json]** et on exécute le projet :

3.7 projet [vuejs-05] : directive [v-for]

Le projet **[vuejs-05]** présente la directive **[v-for]** :



3.7.1 Le script principal [main.js]

Le code du script principal **[main.js]** est identique à celui du script **[main.js]** des projets précédents.

3.7.2 Le composant principal [App]

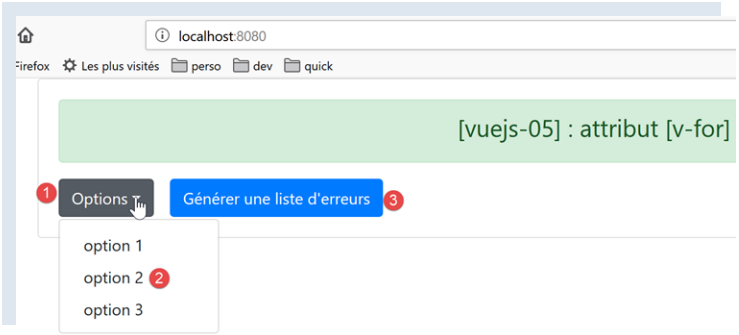
Le code du composant **[App]** est le suivant :

```
1.  <template>
2.    <b-container>
3.      <b-card>
4.        <b-alert show variant="success" align="center">
5.          <h4>[vuejs-05] : attribut [v-for]</h4>
6.        </b-alert>
7.        <VFor />
8.      </b-card>
9.    </b-container>
10. </template>
11.
12.
13. <script>
```

```
14.   import VFor from "./components/VFor.vue";
15.
16.   export default {
17.     name: "app",
18.     components: {
19.       VFor
20.     }
21.   };
22. </script>
```

- lignes 7, 14, 19 : le composant **[App]** utilise le composant **[VFor]** ;

3.7.3 Le composant [VFor]



Le rendu visuel sera le suivant :

Le code du composant **[VFor]** est le suivant :

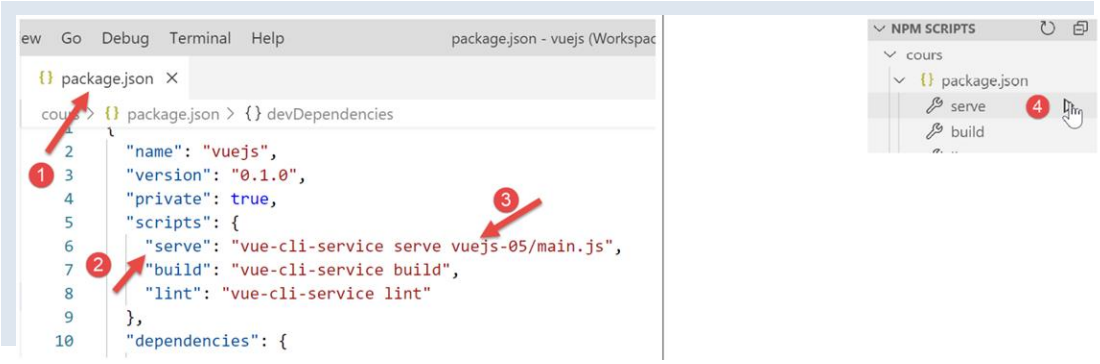
```
1.  <template>
2.    <div>
3.      <!-- une liste déroulante -->
4.      <b-dropdown id="dropdown" text="Options">
5.        <b-dropdown-item v-for="(option,index) in options"
6.                         :key="option.id"
7.                         @click="select(index)">{{option.text}}</b-dropdown-item>
8.      </b-dropdown>
9.      <!-- bouton -->
10.     <b-button class="ml-3"
11.               variant="primary"
12.               @click="generateErrors"
13.               v-if="!error">Générer une liste d'erreurs</b-button>
14.     <!-- alerte -->
15.     <b-alert show variant="danger" v-if="error" class="mt-3">
16.       Les erreurs suivantes se sont produites :
17.       <br />
18.       <ul>
19.         <li v-for="(erreur,index) in erreurs" :key="index">{{erreur}}</li>
20.       </ul>
21.     </b-alert>
22.   </div>
23. </template>
24. <!-- script -->
25. <script>
26.   export default {
```

```
27.     name: "VFor",
28.
29.     // propriétés statiques du composant
30.     data() {
31.       return {
32.         // liste des erreurs
33.         erreurs: [],
34.         // erreur ou pas
35.         error: false,
36.         // liste des options du menu
37.         options: [
38.           { text: "option 1", id: 1 },
39.           { text: "option 2", id: 2 },
40.           { text: "option 3", id: 3 }
41.         ]
42.       };
43.     },
44.
45.     // méthodes
46.     methods: {
47.       // génération d'une liste d'erreurs
48.       generateErrors() {
49.         this.erreurs = ["erreur 1", "erreur 2", "erreur 3"];
50.         this.error = true;
51.       },
52.       // l'utilisateur a sélectionné une option
53.       select(index) {
54.         alert("Vous avez choisi : " + this.options[index].text);
55.       }
56.     }
57.   };
58. </script>
```

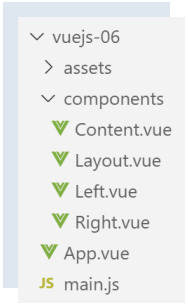
Commentaires

- ligne 4 : la balise <b-dropdown> sert à définir une liste déroulante **[1]** sous la forme d'un bouton qu'on clique pour voir les options de la liste **[2]**. **[text='Options']** définit le texte affiché sur le bouton **[1]** ;
- lignes 5-7 : la balise <dropdown-item> définit un élément de la liste déroulante ;
- ligne 5 : l'attribut **[v-for]** indique que la balise <dropdown-item> doit être répétée pour chaque élément **[option]** de l'attribut **[options]**, lignes 37-41, du composant. **[index]** représente le n° de l'élément dans la liste **[0, 1, ..., n]**. Le nom de l'élément **[option]** ainsi que celui de l'index sont libres. On aurait pu écrire **[<b-dropdown-item v-for="(o,i) in options" :key="o.id" @click="select(i)">{{o.text}}</b-dropdown-item>]** ;
- ligne 6 : si on oublie l'attribut **[key]**, ESLint émet un warning. La valeur de l'attribut **[key]** doit persister dans le temps. Aussi la valeur **[index]** de l'élément ne convient-elle pas. Car si cet élément est supprimé, les valeurs **[index]** de ceux qui sont derrière lui dans la liste vont être décrémentées de 1. Aussi ici, prend-on comme valeur de clé la valeur **[option.id]**, lignes 38-40, qui ne changera pas en cas de suppression d'un élément. L'attribut **[key]** est un élément d'optimisation de régénération du DOM (Document Object Model) par **[Vue.js]** lorsque la liste doit être régénérée. Notez la notation **[:key]**, car **[key]** a une valeur dynamique ;
- ligne 7 : la méthode **[select(index)]**, lignes 49-51, sera appelée lorsque l'utilisateur cliquera sur un élément de la liste ;
- ligne 7 : le texte de l'option sera la valeur **[option.text]** définie aux lignes 37-41 ;
- ligne 10 : le bouton **[3]**. **[class='ml-3]** signifie margin (m) left (l) de trois spacers. **[@click="generateErrors"]** indique que la méthode **[generateErrors]**, lignes 45-48, sera exécutée lors d'un **[click]** sur le bouton. **[v-if="!error"]** indique que l'affichage du bouton est conditionné à la valeur de l'attribut statique **[error]** de la ligne 35 ;
- lignes 15-21 : une alerte de type **[danger] [4]** contrôlée elle-aussi par l'attribut statique **[error]** de la ligne 35. L'attribut **[class='mt-3']** (**m**argin **t**op **3** spacers) fixe l'espace entre cette alerte et l'élément qui est au-dessus de lui ;
- ligne 27 : la balise HTML
 fait un saut de ligne ;
- ligne 18 : début d'une liste non ordonnée **[ul=unordered list]** ;
- ligne 19 : la balise <li> définit un élément de la liste <ul>. Là encore, on utilise une directive **[v-for]** pour générer la balise plusieurs fois. Autant de fois ici que le tableau **[erreurs]** de la ligne 33 aura d'éléments. On utilise ici l'attribut **[:key=index]**. On a dit précédemment que l'index des éléments d'une liste ne faisait pas un bon discriminant entre éléments de la liste car en cas de suppression d'un élément, tous les éléments qui suivent voient leur index changer. Ici ça n'a pas d'importance car les éléments de la liste d'erreurs ne sont pas susceptibles d'être supprimés ;
- ligne 19 : l'élément sert à afficher l'élément **[erreur]** de la liste **[erreurs]** ;
- lignes 30-43 : tous les éléments dynamiques du **[template]** sont des attributs du composant. Il n'y a ici aucune propriété de type **[props]** dont la valeur serait fixée par le composant parent ;
- lignes 48-55 : la méthode **[generateErrors]** génère la liste d'erreurs à afficher par la balise <ul> des lignes 16-18. De plus, elle modifie l'attribut statique **[error]**, à la fois pour afficher cette liste d'erreurs (ligne 15) et cacher le bouton de génération (ligne 13) ;

3.7.4 Exécution du projet



3.8 projet [vuejs-06] : mise en page d'une vue avec des slots



3.8.1 Le script principal [main.js]

Le script principal **[main.js]** reste inchangé.

3.8.2 Le composant principal [App]

Le code du composant **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <b-alert show variant="success" align="center">
5.          <h4>[vuejs-06] : mise en page avec des slots</h4>
6.        </b-alert>
7.        <Content />
8.      </b-card>
9.    </div>
10. </template>
11.
12.
13. <script>
14. import Content from "./components/Content";
15. export default {
16.   name: "app",
17.   // composants utilisés
18.   components: {
19.     Content,
20.   },
21. };
22. </script>
```

Le composant **[App]** introduit un nouveau composant appelé **[Content]** (lignes 7, 14, 19).

3.8.3 Le composant [Layout]

Le composant **[Layout]** sert à définir une mise en page des vues de l'application :

```
1.  <template>
2.    <!-- ligne -->
3.    <b-row>
4.      <!-- zone à trois colonnes -->
5.      <b-col cols="3" v-if="left">
6.        <slot name="slot-left" />
7.      </b-col>
8.      <!-- zone à neuf colonnes -->
9.      <b-col cols="9" v-if="right">
10.       <slot name="slot-right" />
11.     </b-col>
12.   </b-row>
13. </template>
14.
15. <script>
16.   export default {
17.     name: "patron",
18.     // paramètres
19.     props: {
20.       left: {
21.         type: Boolean
22.       },
23.       right: {
24.         type: Boolean
25.       }
26.     }
27.   };
28. </script>
```

Commentaires

- le composant **[Layout]** définit une unique ligne (lignes 3-12). Celle-ci est divisée en deux colonnes :
  - une colonne composée de 3 colonnes Bootstrap (lignes 5-7) ;
  - une colonne composée de 9 colonnes Bootstrap (lignes 9-11) ;
- ligne 5 : la présence de la colonne de gauche (lignes 5-7) est conditionnée par la valeur du paramètre **[left]**, définie dans la propriété **[props]** du composant (lignes 20-22) ;
- ligne 9 : la présence de la colonne de droite (lignes 9-11) est conditionnée par la valeur du paramètre **[right]**, définie dans la propriété **[props]** du composant (lignes 23-25) ;
- les valeurs des paramètres **[left, right]** doivent être fixées par un composant parent du composant **[Layout]** ;
- ligne 6 : on ne fixe pas le contenu de la colonne de gauche. On donne simplement un nom à cette colonne avec la balise <slot>. Ici elle s'appelera **[slot-left]**. Un composant utilisant le composant **[Content]** devra indiquer ce qu'il veut mettre dans la zone appelée **[slot-left]** ;
- ligne 10 : la colonne de droite s'appellera **[slot-right]** ;

3.8.4 Le composant [Right]

Le composant **[Right]** est le suivant :

```
1.  <template>
2.    <!-- un message dans une alerte de type warning -->
3.    <b-alert show variant="warning" align="center">
4.      <h4>{{msg}}</h4>
5.    </b-alert>
6.  </template>
7.
8.  <!-- script -->
9.  <script>
10.   export default {
11.     name: "droite",
12.     // paramètres
13.     props: {
14.       msg: String
15.     }
16.   };
17. </script>
```

- lignes 3-5 : le composant **[Right]** affiche un message dans une alerte de type **[warning]**. Ce message **[msg]** est défini comme étant un paramètre du composant, ligne 14. Le composant parent devra donc lui donner une valeur ;

3.8.5 Le composant [Left]

Le composant **[Left]** est le suivant :

```
1.  <template>
2.    <!-- un message dans une alerte de type primary -->
3.    <b-alert show variant="primary" align="center">
4.      <h4>{{msg}}</h4>
5.    </b-alert>
6.  </template>
7.
8.  <!-- script -->
9.  <script>
10.   export default {
11.     name: "gauche",
12.     // paramètres
13.     props: {
14.       msg: String
15.     }
16.   };
17. </script>
```

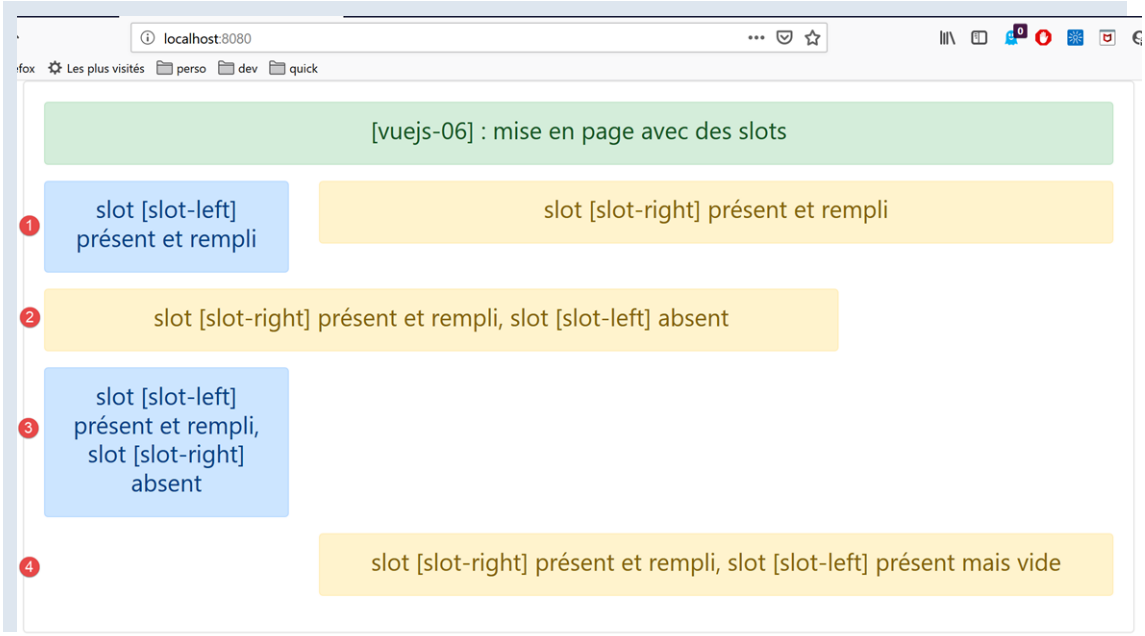
- lignes 3-5 : le composant **[Left]** affiche un message dans une alerte de type **[primary]**. Ce message **[msg]** est défini comme étant un paramètre du composant, ligne 14. Le composant parent devra donc lui donner une valeur ;

3.8.6 Le composant [Content]

On se rappelle que le composant **[Content]** est le composant affiché par la vue principale **[App.vue]** :

```
1.  <template>
2.    <div>
3.      <!-- colonnes gauche et droite remplies -->
4.      <Layout :left="true" :right="true">
5.        <Right slot="slot-right" msg="slot [slot-right] présent et rempli" />
6.        <Left slot="slot-left" msg="slot [slot-left] présent et rempli" />
7.      </Layout>
8.      <!-- colonnes gauche basente. colonne droite remplie -->
9.      <Layout :left="false" :right="true">
10.       <Right slot="slot-right" msg="slot [slot-right] présent et rempli, slot [slot-left] absent" />
11.     </Layout>
12.     <!-- colonnes gauche remplie, colonne droite absente -->
13.     <Layout :left="true" :right="false">
14.       <Left slot="slot-left" msg="slot [slot-left] présent et rempli, slot [slot-right] absent" />
15.     </Layout>
16.     <!-- colonnes gauche présente mais pas remplie, colonne droite remplie -->
17.     <Layout :left="true" :right="true">
18.       <Right slot="slot-right" msg="slot [slot-right] présent et rempli, slot [slot-left] présent mais vide" />
19.     </Layout>
20.   </div>
21. </template>
22.
23. <!-- script -->
24. <script>
25.   import Layout from "./Layout";
26.   import Left from "./Left";
27.   import Right from "./Right";
28.   export default {
29.     name: "contenu",
30.     // composants
31.     components: {
32.       Layout,
33.       Left,
34.       Right
35.     }
36.   };
37. </script>
```

Rendu visuel



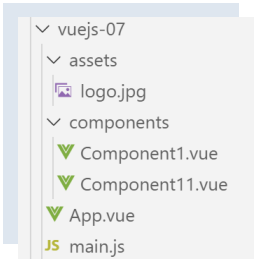
Commentaires

- le composant **[Layout]** est utilisé 4 fois (lignes 4-7, 9-11, 13-15, 17-19). Si on se rappelle que le composant **[Layout]** définit 1 ligne, le **[template]** ci-dessus définit quatre lignes. On se rappelle également que la ligne du **[Layout]** définit deux colonnes :
  - une colonne appelée **[slot-left]** qui occupe les 3 colonnes Bootstrap de gauche ;
  - une colonne appelée **[slot-Right]** qui occupe les 9 colonnes Bootstrap de droite ;
- lignes 4-7 : définit une ligne **[1]** où le composant **[Left]** occupe la colonne **[slot-left]** et le composant **[Right]** la colonne **[slot-right]** ;
- lignes 9-11 : définit une ligne **[2]** où la colonne **[slot-left]** n'est pas affichée et le composant **[Right]** occupe la colonne **[slot-right]** ;
- lignes 13-15 : définit une ligne **[3]** où la colonne **[slot-right]** n'est pas affichée et le composant **[Left]** occupe la colonne **[slot-left]** ;
- lignes 17-19 : définit une ligne **[4]** où la colonne **[slot-left]** est affichée **[:left='true']** mais pas remplie et le composant **[Right]** occupe la colonne **[slot-right]** ;

Au final, le composant **[Layout]** a servi de mise en page au composant **[Content]**.

3.9 projet [vuejs-07] : remontée d'événements dans la hiérarchie des composants

L'arborescence du projet est la suivante :



3.9.1 Le script principal [main.js]

Le script principal **[main.js]** reste inchangé.

3.9.2 Le composant principal [App]

Le code du composant **[App]** est le suivant :

```
1.  <template>
```

```
2.    <div class="container">
3.      <b-card>
4.        <b-alert show variant="success" align="center">
5.          <h4>[vuejs-07] : remontée d'événements dans la hiérarchie des composants</h4>
6.        </b-alert>
7.        <Component1 />
8.      </b-card>
9.    </div>
10. </template>
11.
12. <script>
13.   import Component1 from './components/Component1'
14.   export default {
15.     name: "app",
16.     components: {
17.       Component1
18.     }
19.   };
20. </script>
```

Le composant **[App]** utilise un nouveau composant **[Component1]** (lignes 7, 13, 17).

3.9.3 Le composant [Component11]

Le code du composant **[Component11]** est le suivant :

```
1.  <template>
2.    <b-button @click="createEvent">Créer un événement</b-button>
3.  </template>
4.  <!-- script -->
5.  <script>
6.    export default {
7.      name: "component11",
8.      // méthodes du composant
9.      methods: {
10.       // gestion de l'évt [click] sur le bouton
11.       createEvent() {
12.         // on émet un événement couplé à une donnée
13.         this.$emit("someEvent", { x: 2, y: 4 })
14.       }
15.     }
16.   };
17. </script>
```

Commentaires

- lignes 1-3 : le composant **[Component11]** ne comporte qu'un bouton qu'on peut cliquer. Lorsqu'on le clique, la méthode **[createEvent]** des lignes 11-14 est exécutée ;
- ligne 13 : chaque instance **[Vue]** dispose d'une méthode **[$emit]** qui permet d'émettre un événement :
  - le 1er paramètre est le nom de l'événement émis ;
  - le second paramètre est la donnée que l'on veut associer à cet événement ;
  - l'événement émis remonte la hiérarchie des composants qui chapeautent le composant **[Component11]**. Il remonte la hiérarchie jusqu'à ce qu'il trouve un composant disposé à le traiter. Cette remontée commence par le composant parent de **[Component11]** ;

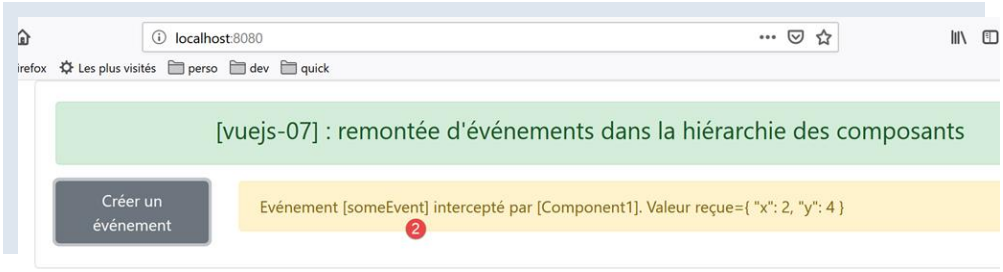
3.9.4 Le composant [Component1]

Le code du composant **[Component1]** est le suivant :

```
1.  <template>
2.    <b-row>
3.      <!-- le composant qui lance l'événement -->
4.      <b-col cols="2">
5.        <Component11 @someEvent="doSomething" />
6.      </b-col>
7.      <!-- message affiché par la méthode de gestion de l'évt-->
8.      <b-col>
9.        <b-alert show
10.                variant="warning"
11.                v-if="showMsg">Evénement [someEvent] intercepté par [Component1]. Valeur reçue={{data}}</b-alert>
12.     </b-col>
13.   </b-row>
14. </template>
15.
16. <script>
```

```
17.    import Component11 from "./Component11";
18.    export default {
19.      name: "component1",
20.      // composants
21.      components: {
22.        Component11
23.      },
24.      // état du composant
25.      data() {
26.        return {
27.          data: "",
28.          showMsg: false
29.        };
30.      },
31.      // méthodes de gestion des évts
32.      methods: {
33.        doSomething(data) {
34.          // data est l'objet qui a été associé à l'évt [someEvent]
35.          this.data = data;
36.          // affiche l'alerte
37.          this.showMsg = true;
38.        }
39.      }
40.    };
41.  </script>
```

Rendu visuel



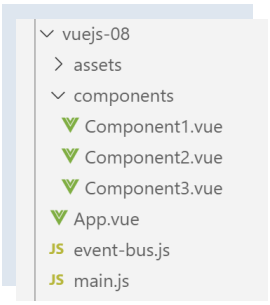
Commentaires

- ligne 5 : **[Component1]** est parent de **[Component11]** et peut donc 'écouter' (c'est le terme) les événements émis par ce composant. On sait que celui-ci peut émettre l'événement **[someEvent]**. L'attribut **[@someEvent="doSomething"]** indique que si l'événement **[someEvent]** émis par **[Component11]** se produit, alors la méthode **[doSomething]** de la ligne 33 doit être exécutée ;
- lignes 9-11 : un message affiché par la méthode **[doSomething]**. Son affichage est contrôlé par le booléen **[showMsg]** de la ligne 28. L'alerte affiche l'attribut **[data]** de la ligne 27 ;
- ligne 33 : la méthode **[doSomething]** exécutée lorsque l'événement **[someEvent]** se produit ligne 5, reçoit comme paramètre la donnée **[data]** associée à cet événement. Ce paramètre est affecté à l'attribut **[data]** de la ligne 27 et qui est affiché par la ligne 11 ;
- ligne 37 : on met l'attribut **[showMsg]** à **[true]** pour afficher l'alerte des lignes 9-11 ;

3.9.5 Exécution du projet

3.10 projet [vuejs-08] : événements indépendants de la hiérarchie des composants, cycle de vie des composants

Le projet **[vuejs-08]** lontre comment des composants non reliés directement dans la hiérarchie des composants peuvent néanmoins communiquer via des événements. L'arborescence du projet **[vuejs-08]** est la suivante :



3.10.1 Le script principal [main.js]

Le script **[main.js]** reste ce qu'il était dans les exemples précédents.

3.10.2 Le script [even-bus.js]

Le script **[event-bus.js]** est l'outil que nous allons utiliser pour émettre et écouter des événements :

```
1.  import Vue from 'vue';
2.  const eventBus = new Vue();
3.  export default eventBus;
```

Le script **[event-bus]** se contente de créer une instance de la classe **[Vue]** (lignes 1-2) et de l'exporter (ligne 3). La classe **[Vue]** possède deux méthodes pour gérer les événements :

- **[Vue].$emit(nom, donnée)** : permet d'émettre un événement nommé **[nom]** et associé à la donnée **[donnée]** ;
- **[Vue].$on(nom, fn)** : permet d'intercepter l'événement nommé **[nom]** et de le faire traiter par la fonction **[fn]**. La fonction **[fn]** recevra en paramètre la donnée **[donnée]** associée à l'événement par son émetteur ;

Cette gestion d'événements n'est pas liée à la hiérarchie des composants. Il faut simplement que les composants qui veulent communiquer ainsi utilisent **la même instance** de la classe **[Vue]** pour émettre / écouter les événements. Dans notre exemple, nous utiliserons l'instance **[eventBus]** définie par le script **[event-bus.js]** précédent.

3.10.3 La vue principale [App.vue]

Le code de la vue principale **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <b-alert show variant="success" align="center">
5.          <h4>[vuejs-08] : événements indépendants de la hiérarchie des composants, cycle de vie des composants</h4>
6.        </b-alert>
```

```
7.        <!-- les trois composants qui communiquent par évt -->
8.        <Component1 />
9.        <Component3 />
10.       <Component2 />
11.     </b-card>
12.   </div>
13. </template>
14.
15. <script>
16.   import Component1 from "./components/Component1";
17.   import Component2 from "./components/Component2";
18.   import Component3 from "./components/Component3";
19.   export default {
20.     name: "app",
21.     // composants
22.     components: {
23.       Component1,
24.       Component2,
25.       Component3
26.     }
27.   };
28. </script>
```

- lignes 8-10 : la vue principale **[App]** utilise trois composants **[Component1, Component2, Component3]** qui vont communiquer par événements. Les trois composants sont au même niveau dans la vue **[App]**. De plus, il n'y aura pas de relation de hiérarchie entre eux ;

3.10.4 Le composant [Component1]

Le code du composant **[Component1]** est le suivant :

```
1.  <template>
2.    <b-row>
3.      <b-col>
4.        <b-alert show
5.                 variant="warning"
6.                 v-if="showMsg">Evénement [someEvent] intercepté par [Component1]. Valeur reçue={{data}}</b-alert>
7.      </b-col>
8.    </b-row>
9.  </template>
10.
11. <script>
12.   import eventBus from "../event-bus.js"
13.   export default {
14.     name: "component1",
15.     // état du composant
16.     data() {
17.       return {
18.         data: "",
19.         showMsg: false
20.       };
21.     },
22.     // méthodes de gestion des évts
23.     methods: {
24.       // gestion de l'evt [someEvent]
25.       doSomething(data) {
26.         this.data = data;
27.         this.showMsg = true;
28.       }
29.     },
30.     // gestion du cycle de vie du composant
31.     // évt [created] - le composant a été créé
32.     created() {
33.       // écoute de l'évt [someEvent]
34.       eventBus.$on("someEvent", this.doSomething);
35.     }
36.   };
37. </script>
```

Commentaires

- lignes 4-6 : une alerte qui affiche la donnée **[data]** de la ligne 18. Cette donnée sera initialisée par le gestionnaire d'événement **[doSomething]** de la ligne 25. Ce gestionnaire est activé à réception de l'événement **[someEvent]** (ligne 34). L'alerte est affichée conditionnellement à la valeur de l'attribut **[showMsg]** de la ligne 19. Cet attribut est lui également positionné par le gestionnaire d'événement **[doSomething]** de la ligne 25 ;
- ligne 32 : la fonction **[created]** est un gestionnaire d'événement. Elle gère l'événement **[created]**, un événement émis au cours du cycle de vie du composant. Il en existe d'autres **[beforeCreate, created, beforeMount, mounted, beforeUpdate, updated, beforeDestroy, destroyed]**. L'événement **[created]** est émis lorsque le composant a été créé ;
- ligne 12 : l'instance **[eventBus]** de la classe **[Vue]** est importée ;

- ligne 34 : elle est utilisée pour écouter l'événement **[someEvent]**. Lorsque celui-ci se produit, la méthode **[doSomething]** de la ligne 25 est appelée ;
- ligne 25 : la méthode **[doSomething]** reçoit comme paramètre **[data]** la donnée que l'émetteur de l'événement **[someEvent]** a associée à l'événement ;
- lignes 26-27 : l'état du composant est modifié pour que l'alerte des lignes 4-6 affiche la donnée reçue ;

3.10.5 Le composant [Component2]

Le composant **[Component2]** est le suivant :

```
1.  <template>
2.    <div>
3.      <b-button @click="createEvent">Créer un événement</b-button>
4.    </div>
5.  </template>
6.  <!-- script -->
7.  <script>
8.    import eventBus from '../event-bus.js'
9.    export default {
10.     name: "component2",
11.     // méthodes de gestion des évts
12.     methods: {
13.       createEvent() {
14.         eventBus.$emit("someEvent", { x: 2, y: 4 })
15.       }
16.     }
17.   };
18. </script>
```

Commentaires

- ligne 3 : un bouton pour créer un événement. Lorsque l'utilisateur clique sur ce bouton, la méthode **[createEvent]** de la ligne 13 est appelée ;
- ligne 8 : l'instance **[eventBus]** définie par le script **[event-bus.js]** est importée ;
- ligne 14 : cette instance est utilisée pour émettre un événement nommé **[someEvent]** associé à la donnée **[{ x: 2, y: 4 }]**. Au final, lorsque l'utilisateur clique sur le bouton de la ligne 3, l'événement **[someEvent]** est émis. Si on se rappelle la définition de **[Component1]**, celui-ci interceptera cet événement ;

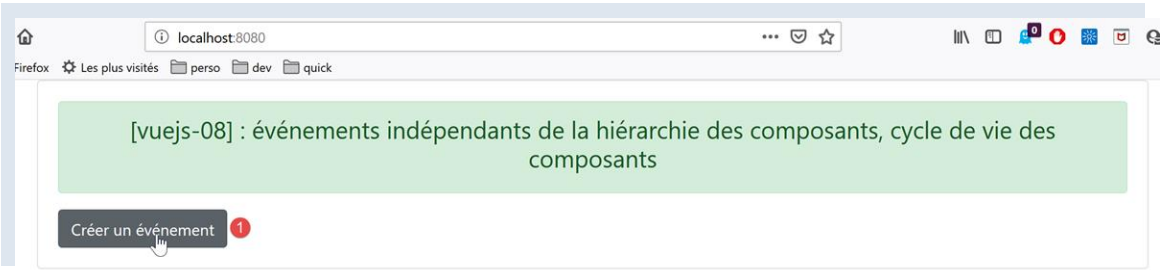
3.10.6 Le composant [Component3]

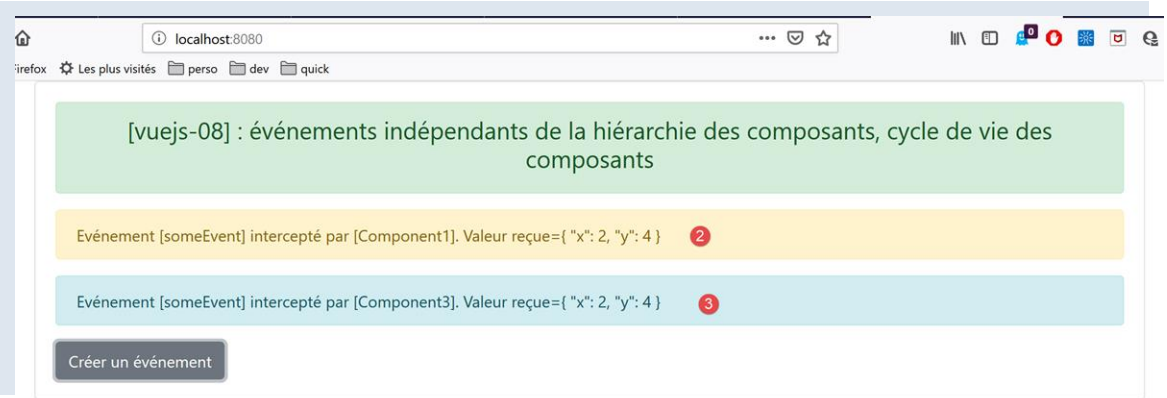
Le code de ce composant est le suivant :

```
1.  <template>
2.    <b-row>
3.      <b-col>
4.        <b-alert show
5.                 v-if="showMsg">Evénement [someEvent] intercepté par [Component3]. Valeur reçue={{data}}</b-alert>
6.      </b-col>
7.    </b-row>
8.  </template>
9.
10. <script>
11.   import eventBus from "../event-bus.js"
12.   export default {
13.     name: "component3",
14.     // état du composant
15.     data() {
16.       return {
17.         data: "",
18.         showMsg: false
19.       };
20.     },
21.     methods: {
22.       // gestion de l'evt [someEvent]
23.       doSomething(data) {
24.         this.data = data;
25.         this.showMsg = true;
26.       }
27.     },
28.     // gestion du cycle de vie du composant
29.     // évt [created] - le composant a été créé
30.     created() {
31.       // écoute de l'évt [someEvent]
32.       eventBus.$on("someEvent", this.doSomething);
33.     }
34.   };
35. </script>
```

[Component3] est un clone de **[Component1]**. Il n'est là que pour montrer qu'un événement peut être intercepté par plusieurs composants.

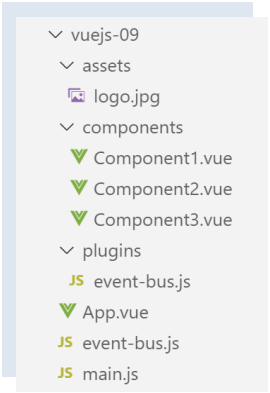
3.10.7 Exécution du projet [vuejs-08]





3.11 projet [vuejs-09] : utilisation d'un plugin [eventBus]

Le projet **[vuejs-09]** est identique au projet **[vuejs-08]** si ce n'est qu'il introduit la notion de plugin. L'arborescence du projet est la suivante :



3.11.1 Le plugin [./plugins/event-bus]

Le script **[./event-bus.js]** reste ce qu'il était dans l'exemple précédent :

```
1.  import Vue from 'vue';
2.  const eventBus = new Vue();
3.  export default eventBus;
```

Le plugin **[./plugins/event-bus.js]** est le suivant :

```
1.  export default {
2.    install(Vue, eventBus) {
3.      // ajoute une propriété [$eventBus] à la classe Vue
4.      Object.defineProperty(Vue.prototype, '$eventBus', {
5.        // lorsque Vue.$eventBus est référencé, on rend le 2ième paramètre [eventBus]
6.        get: () => eventBus,
7.      })
8.    }
9.  }
```

Commentaires

- un plugin **[Vue]** est un objet ayant une fonction **[install]** (ligne 2). Celle-ci sera automatiquement appelée lorsqu'un code déclare l'utilisation du plugin ;
- lignes 1-9 : l'objet exporté par le script ;
- ligne 2 : la fonction **[install]** accepte ici deux paramètres :
  - **[Vue]** : la fonction obtenue par l'instruction **[import Vue from 'Vue']**. Peut-être assimilée à une classe ;
  - **[eventBus]** : l'objet exporté par le script **[./event-bus]** ;
- lignes 4-7 : modifient la définition (on dit le prototype) de la fonction **[Vue]** en lui ajoutant la propriété **[$eventBus]**. Si on raisonne avec le terme **[classe]**, la propriété **[$eventBus]** est ajoutée à la clase **[Vue]**. Les composants qui sont des instances de **[Vue]** auront donc accès à cette nouvelle propriété ;
- la ligne 6 indique que lorsqu'on référencera la propriété **[Vue].$eventBus**, on obtiendra le paramètre **[eventBus]** de la ligne 2. Nous allons voir un peu plus loin que ce second paramètre sera l'objet **[eventBus]** exporté par le script **[./event-bus.js]**. Donc au final, lorsqu'un composant C utilisera l'expression **[C.$eventBus]** il référencera l'objet **[eventBus]** exporté par le script **[./event-bus.js]**. Cela lui évitera d'importer le script **[./event-bus.js]**. L'intérêt du plugin est là : simplifier l'accès à l'objet **[eventBus]** exporté par le script **[./event-bus.js]** ;
- on notera que le plugin n'a pas à s'appeler lui-même **[event-bus.js]**. On aurait pu l'appeler **[plugin-event-bus]** par exemple ;
- on notera également que le terme $ dans **[$eventBus]** est une convention pour désigner les propriétés de **[Vue]** qui ont été ajoutées via des plugins. On n'est pas obligés d'observer cette convention. Dans ce texte, nous l'observerons ;

3.11.2 Le script principal [main.js]

Le script **[./plugins/event-bus.js]** définit un plugin pour le framework **[Vue.js]**. Ce plugin n'est pas encore utilisé, juste défini. C'est le script **[main.js]** qui l'active :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // plugins
6.  import BootstrapVue from 'bootstrap-vue'
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import 'bootstrap/dist/css/bootstrap.css'
11. import 'bootstrap-vue/dist/bootstrap-vue.css'
12.
13. // eventbus
14. import eventBus from './event-bus';
15. import PluginEventBus from './plugins/event-bus';
16. Vue.use(PluginEventBus, eventBus);
17.
18. // configuration
19. Vue.config.productionTip = false
20.
21. // instanciation projet [App]
22. new Vue({
23.   render: h => h(App),
24. }).$mount('#app')
```

Commentaires

- les lignes 14-16 activent le plugin **[PluginEventBus]**. Après la ligne 16, toutes les instances de la classe (fonction) **[Vue]** possèdent la propriété **[$eventBus]** qui pointe pour chacune d'elles sur le même object exporté par le script **[./event-bus.js]**. Ce sera le cas pour chacun des composants du projet ;

3.11.3 La vue principale [App]

La vue principale **[App]** reste ce qu'elle était dans le projet précédent.

3.11.4 Le composant [Component1]

Le composant **[Component1]** utilise désormais sa propriété **[$eventBus]** pour écouter l'événement **[someEvent]** :

```
1.  <template>
2.    <b-row>
3.      <b-col>
4.        <b-alert show
5.                 variant="warning"
6.                 v-if="showMsg">Evénement [someEvent] intercepté par [Component1]. Valeur reçue={{data}}</b-alert>
7.      </b-col>
8.    </b-row>
9.  </template>
10.
11. <script>
12.   export default {
13.     name: "component1",
14.     // état du composant
15.     data() {
16.       return {
17.         data: "",
18.         showMsg: false
19.       };
20.     },
21.     // méthodes de gestion des évts
22.     methods: {
23.       // gestion de l'evt [someEvent]
24.       doSomething(data) {
25.         this.data = data;
26.         this.showMsg = true;
27.       }
28.     },
29.     // gestion du cycle de vie du composant
30.     // évt [created] - le composant a été créé
31.     created() {
32.       // écoute de l'évt [someEvent]
33.       this.$eventBus.$on("someEvent", this.doSomething);
34.     }
35.   };
36. </script>
```

Commentaires

- ligne 33, utilisation de la propriété **[this.$eventBus]** du composant. On remarquera de plus que le script ligne 11 n'importe plus le script **[./event-bus.js]** ;

3.11.5 Le composant [Component2]

Le composant **[Component2]** utilise désormais sa propriété **[$eventBus]** pour émettre l'événement **[someEvent]** :

```
1.  <template>
2.    <div>
3.      <b-button @click="createEvent">Créer un événement</b-button>
4.    </div>
5.  </template>
6.  <!-- script -->
7.  <script>
8.    export default {
9.      name: "component2",
10.     // méthodes de gestion des évts
11.     methods: {
12.       createEvent() {
13.         this.$eventBus.$emit("someEvent", { x: 2, y: 4 })
14.       }
15.     }
```

```
16.   };
17. </script>
```

Commentaires

- ligne 13, utilisation de la propriété **[this.$eventBus]** du composant. On remarquera de plus que le script ligne 7 n'importe plus le script **[./event-bus.js]** ;

3.11.6 Composant [Component3]

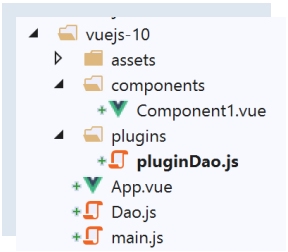
Le composant **[Component3]** a le même code que **[Component1]**. Lui également écoute l'événement **[someEvent]**.

3.11.7 Exécution du projet

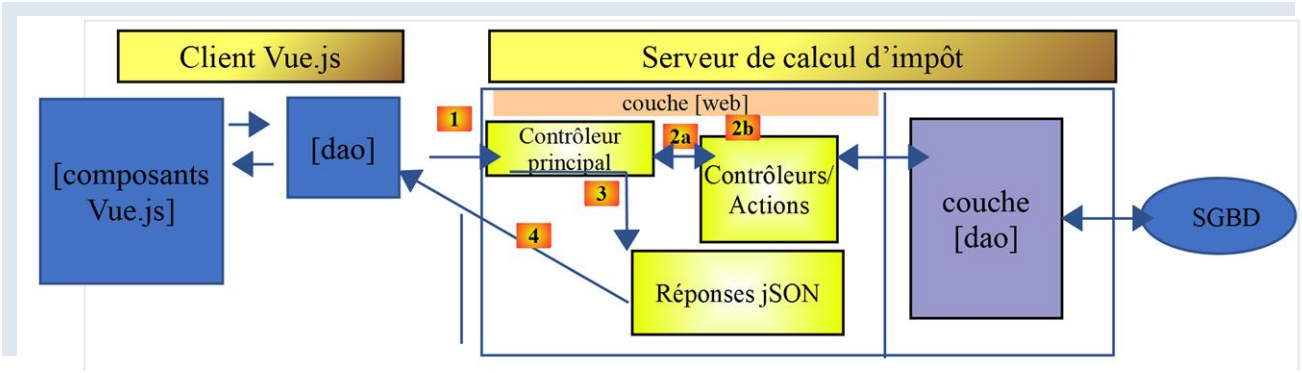
On obtient les mêmes résultats que dans le projet précédent.

3.12 projet [vuejs-10] : plugin [dao], requêtes HTTP asynchrones

L'arborescence du projet **[vuejs-10]** est la suivante :



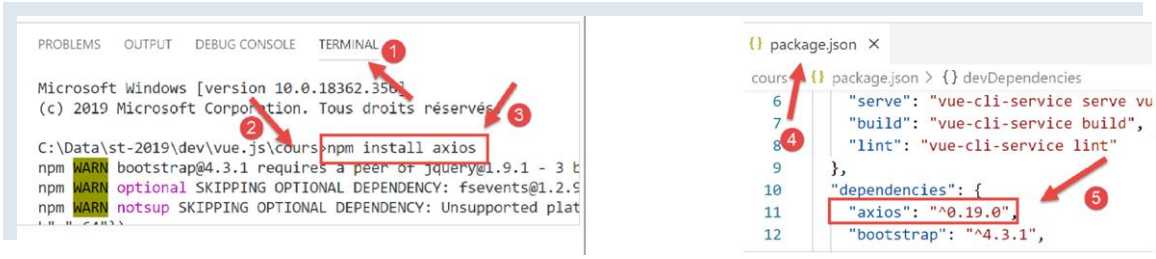
Le projet **[vuejs-10]** montre un composant faisant une requête HTTP à un serveur distant. L'architecture utilisée est la suivante :



Un composant **[Vue.js]** utilise la couche **[dao]** pour dialoguer avec le serveur de calcul de l'impôt.

3.12.1 Installation des dépendances

L'application **[vuejs-10]** utilise la bibliothèque **[axios]** pour faire les requêtes asynchrones vers le serveur de calcul d'impôt. Il nous faut installer cette dépendance :



- en **[4-5]**, la ligne ajoutée au fichier **[package.json]** après l'installation de la bibliothèque **[axios]** **[1-3]** ;

3.12.2 La classe [Dao]

La classe **[Dao]** est celle qui a été développée au paragraphe **[La classe Dao]**. Nous la redonnons ici pour mémoire :

```
1.  'use strict';
2.
3.  // imports
4.  import qs from 'qs'
5.
6.  // classe [Dao]
7.  class Dao {
8.
9.    // constructeur
10.   constructor(axios) {
11.     this.axios = axios;
12.     // cookie de session
13.     this.sessionCookieName = "PHPSESSID";
14.     this.sessionCookie = '';
15.   }
16.
17.   // init session
18.   async  initSession() {
19.     // options de la requête HHTP [get /main.php?action=init-session&type=json]
20.     const options = {
21.       method: "GET",
22.       // paramètres de l'URL
23.       params: {
24.         action: 'init-session',
25.         type: 'json'
26.       }
27.     };
28.     // exécution de la requête HTTP
29.     return await this.getRemoteData(options);
30.   }
31.
32.   async  authentifierUtilisateur(user, password) {
33.     // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
34.     const options = {
35.       method: "POST",
36.       headers: {
37.         'Content-type': 'application/x-www-form-urlencoded',
38.       },
39.       // corps du POST
40.       data: qs.stringify({
41.         user: user,
42.         password: password
43.       }),
44.       // paramètres de l'URL
45.       params: {
46.         action: 'authentifier-utilisateur'
47.       }
48.     };
49.     // exécution de la requête HTTP
50.     return await this.getRemoteData(options);
51.   }
52.
53.   async getAdminData() {
```

```
54.     // options de la requête HHTP  [get /main.php?action=get-admindata]
55.     const options = {
56.       method: "GET",
57.       // paramètres de l'URL
58.       params: {
59.         action: 'get-admindata'
60.       }
61.     };
62.     // exécution de la requête HTTP
63.     const data = await this.getRemoteData(options);
64.     // résultat
65.     return data;
66.   }
67.
68.   async  getRemoteData(options) {
69.     // pour le cookie de session
70.     if (!options.headers) {
71.       options.headers = {};
72.     }
73.     options.headers.Cookie = this.sessionCookie;
74.     // exécution de la requête HTTP
75.     let response;
76.     try {
77.       // requête asynchrone
78.       response = await this.axios.request('main.php', options);
79.     } catch (error) {
80.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
81.       if (error.response) {
82.         // la réponse du serveur est dans [error.response]
83.         response = error.response;
84.       } else {
85.         // on relance l'erreur
86.         throw error;
87.       }
88.     }
89.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
90.     // on récupère le cookie de session s'il existe
91.     const setCookie = response.headers['set-cookie'];
92.     if (setCookie) {
93.       // setCookie est un tableau
94.       // on cherche le cookie de session dans ce tableau
95.       let trouvé = false;
96.       let i = 0;
97.       while (!trouvé && i < setCookie.length) {
98.         // on cherche le cookie de session
99.         const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
100.        if (results) {
101.          // on mémorise le cookie de session
102.          // eslint-disable-next-line require-atomic-updates
103.          this.sessionCookie = results[1];
104.          // on a trouvé
105.          trouvé = true;
106.        } else {
107.          // élément suivant
108.          i++;
109.        }
110.      }
111.    }
112.    // la réponse du serveur est dans [response.data]
113.    return response.data;
114.  }
115.}
116.
117.// export de la classe
118.export default Dao;
```

Le projet **[vuejs-10]** n'utilise que la méthode asynchrone **[initSession]** des lignes 18-30. On rappelle que la classe **[Dao]** est instanciée avec un paramètre **[axios]**, ligne 10, paramètre initialisé par le code appelant. Ce code appelant sera ici le script **[./main.js]**.

3.12.3 Le plugin [pluginDao]

Le plugin **[pluginDao]** est le suivant :

```
1.  export default {
2.    install(Vue, dao) {
3.      // ajoute une propriété [$dao] à la classe Vue
```

```
4.      Object.defineProperty(Vue.prototype, '$dao', {
5.        // lorsque Vue.$dao est référencé, on rend le 2ième paramètre [dao]
6.        get: () => dao,
7.      })
8.    }
9.  }
```

Si on se souvient de l'explication donnée pour le plugin **[event-bus]**, on voit que le plugin **[pluginDao]** crée dans la classe / fonction **[Vue]**, une nouvelle propriété appelée **[$dao]**. Cette propriété aura (ça reste à montrer) pour valeur, l'objet exporté par le script **[./Dao]**, ç-à-d la classe **[Dao]** précédente.

3.12.4 Le script principal [main.js]

Le code du script principal **[main.js]** est le suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.  import axios from 'axios';
5.
6.  // plugins
7.  import BootstrapVue from 'bootstrap-vue'
8.  Vue.use(BootstrapVue);
9.
10. // bootstrap
11. import 'bootstrap/dist/css/bootstrap.css'
12. import 'bootstrap-vue/dist/bootstrap-vue.css'
13.
14. // couche [dao]
15. import Dao from './Dao';
16. // configuration axios
17. axios.defaults.timeout = 2000;
18. axios.defaults.baseURL = 'http://localhost/php7/scripts-web/impots/version-14';
19. axios.defaults.withCredentials = true;
20. // instanciation couche [dao]
21. const dao = new Dao(axios);
22.
23. // plugin [dao]
24. import pluginDao from './plugins/pluginDao'
25. Vue.use(pluginDao, dao)
26.
27. // configuration
28. Vue.config.productionTip = false
29.
30. // instanciation projet [App]
31. new Vue({
32.   render: h => h(App),
33. }).$mount('#app')
```

Le script **[main.js]** :

- instancie la couche **[dao]** aux lignes 14-21 ;
- intègre le plugin **[pluginDao]** aux lignes 24-25 ;
- ligne 15 : la classe **[Dao]** est importée ;
- lignes 17-18 : on configure l'objet **[axios]** qui réalise les requêtes HTTP. Cet objet est importé à la ligne 4 ;
  - ligne 17 : définition d'un **[timeout]** de 2 secondes ;
  - ligne 18 : l'URL du serveur de calcul de l'impôt ;
  - ligne 19 : pour pouvoir échanger des cookies avec le serveur ;
- lignes 24-25 : utilisation du plugin **[pluginDao]**
  - ligne 24 : import du plugin ;
  - ligne 25 : intégration du plugin. On voit que le second paramètre de la méthode **[Vue.use]** est la référence de la couche **[dao]** définie ligne 21. C'est pour cette raison que la propriété **[Vue.$dao]** désignera la couche **[dao]** dans toutes les instances de la classe / fonction **[Vue]**, ç-à-d dans tous les composants **[Vue.js]** ;

3.12.5 La vue principale [App.vue]

Le code de la vue principale **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
```

```
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-10] : plugin [dao], requêtes HTTP asynchrones</h4>
7.        </b-alert>
8.        <!-- composant faisant une requête asynchrone au serveur de calcul d'impôt-->
9.        <Component1 @error="doSomethingWithError" @endWaiting="endWaiting" @beginWaiting="beginWaiting" />
10.       <!-- affichage d'une éventuelle erreur -->
11.       <b-alert show
12.                variant="danger"
13.                v-if="showError">Evénement [error] intercepté par [App]. Valeur reçue = {{error}}</b-alert>
14.       <!-- message d'attente avec un spinner -->
15.       <b-alert show v-if="showWaiting" variant="light">
16.         <strong>Requête au serveur de calcul d'impôt en cours...</strong>
17.         <b-spinner variant="primary" label="Spinning"></b-spinner>
18.       </b-alert>
19.     </b-card>
20.   </div>
21. </template>
22.
23. <script>
24.   import Component1 from "./components/Component1";
25.   export default {
26.     name: "app",
27.     // état du composant
28.     data() {
29.       return {
30.         // contrôle le spinner d'attente
31.         showWaiting: false,
32.         // contrôle l'affichage de l'erreur
33.         showError: false,
34.         // l'erreur interceptée
35.         error: {}
36.       };
37.     },
38.     // composants utilisés
39.     components: {
40.       Component1
41.     },
42.     // méthodes de gestion des évts
43.     methods: {
44.       // début attente
45.       beginWaiting() {
46.         // on affiche l'attente
47.         this.showWaiting = true;
48.         // on cache le msg d'erreur
49.         this.showError = false;
50.       },
51.       // fin attente
52.       endWaiting() {
53.         // on cache l'attente
54.         this.showWaiting = false;
55.       },
56.       // gestion d'erreur
57.       doSomethingWithError(error) {
58.         // on note qu'il y a eu erreur
59.         this.error = error;
60.         // on affiche le msg d'erreur
61.         this.showError = true;
62.       }
63.     }
64.   };
65. </script>
```

Commentaires

- ligne 9 : **[Component1]** est le composant qui fait la requête HTTP asynchrone. Il peut émettre trois événements :
  - **[beginWaiting]** : la requête va être faite. Il faut afficher un message d'attente à destination de l'utilisateur ;
  - **[endWaiting]** : la requête est terminée. Il faut arrêter l'attente ;
  - **[error]** : la requête s'est mal passée. Il faut afficher un message d'erreur ;
- lignes 10-13 : l'alerte qui affiche l'éventuel message d'erreur. Elle est contrôlée par le booléen **[showError]** de la ligne 33. Elle affiche l'erreur de la ligne 35 ;
- lignes 14-18 : l'alerte qui affiche le message d'attente avec un spinner. Elle est contrôlée par le booléen **[showWaiting]** de la ligne 47 ;
- lignes 45-50 : **[beginWaiting]** est la méthode exécutée à réception de l'événement **[beginWaiting]**. Elle affiche le message d'attente (ligne 47) et cache le message d'erreur (ligne 49) au cas où celui-ci serait visible suite à une opération précédente ;
- lignes 52-55 : **[endWaiting]** est la méthode exécutée à réception de l'événement **[endWaiting]**. Elle cache le message d'attente (ligne 54) ;
- lignes 57-62 : **[doSomethingWithError]** est la méthode exécutée à réception de l'événement **[error]**. Elle enregistre l'erreur reçue (ligne 59) et affiche le message d'erreur (ligne 61) ;

3.12.6 Le composant `[Component1]`

Le code du composant **`[Component1]`** est le suivant :

```
1.  <template>
2.    <b-row>
3.      <b-col>
4.        <b-alert show
5.                 variant="warning"
6.                 v-if="showMsg">Valeur reçue du serveur = {{data}}</b-alert>
7.      </b-col>
8.    </b-row>
9.  </template>
10.
11. <script>
12.   export default {
13.     name: "component1",
14.     // état du composant
15.     data() {
16.       return {
17.         showMsg: false
18.       };
19.     },
20.     // méthodes de gestion des évts
21.     methods: {
22.       // traitement de la donnée reçue du serveur
23.       doSomethingWithData(data) {
24.         // on enregistre la donnée reçue
25.         this.data = data;
26.         // on l'affiche
27.         this.showMsg = true;
28.       }
29.     },
30.     // le composant vient d'être créé
31.     created() {
32.       // on initialise la session avec le serveur - requête asynchrone
33.       // on utilise la promesse rendue par les méthodes de la couche [dao]
34.       // on signale le début de l'opération
35.       this.$emit("beginWaiting");
36.       // on lance l'opération asynchrone
37.       this.$dao
38.         // il s'agit d'initialiser une session jSON avec le serveur de calcul de l'impôt
39.         .initSession()
40.         // méthode qui traite la donnée reçue en cas de succès
41.         .then(data => {
42.           // on traite la donnée reçue
43.           this.doSomethingWithData(data);
44.         })
45.         // méthode qui traite l'erreur en cas d'erreur
46.         .catch(error => {
47.           // on remonte l'erreur au composant parent
48.           this.$emit("error", error.message);
49.         }).finally(() => {
50.           // fin de l'attente
51.           this.$emit("endWaiting");
52.         })
53.     }
54.   };
55. </script>
```

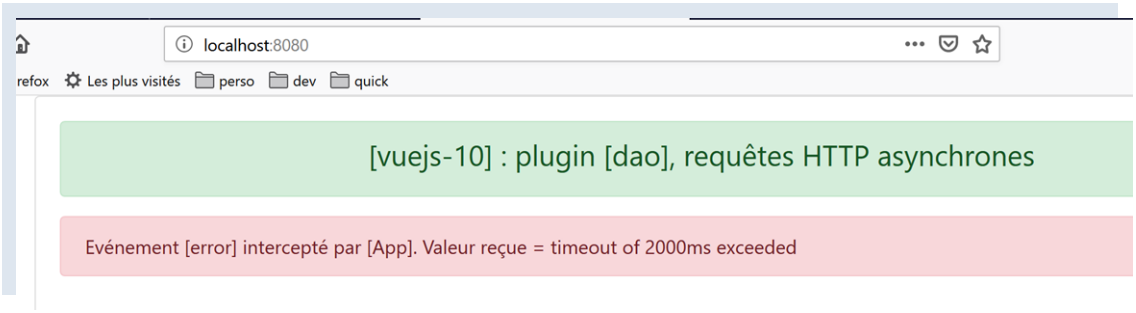
Commentaires

- lignes 4-6 : le composant est constitué d'une unique alerte qui affiche la valeur renvoyée par le serveur de calcul de l'impôt, ceci uniquement en cas de succès de la requête HTTP. Cette alerte est contrôlée par le booléen **`[showMsg]`** de la ligne 17 ;
- lignes 31-53 : la requête HTTP est faite dès que le composant a été créé. On met donc son code dans la méthode **`[created]`** de la ligne 31 ;
- ligne 35 : on indique au composant parent que la requête asynchrone va démarrer ;
- lignes 37-39 : la méthode **`[this.$dao.initSession]`** est exécutée. Elle initialise une session jSON avec le serveur de calcul d'impôt. Le résultat immédiat de cette méthode est une **`[Promise]`** ;
- lignes 41-44 : ce code s'exécute lorsque le serveur a rendu son résultat sans erreur. Le résultat du serveur est dans **`[data]`**. Ligne 43, on demande à la méthode **`[doSomethingWithData]`** de traiter ce résultat ;
- lignes 46-49 : ce code s'exécute en cas d'erreur lors de l'exécution de la requête. Ligne 48, on indique au composant parent qu'une erreur est survenue et on lui passe le message de l'erreur **`[error.message]`** ;

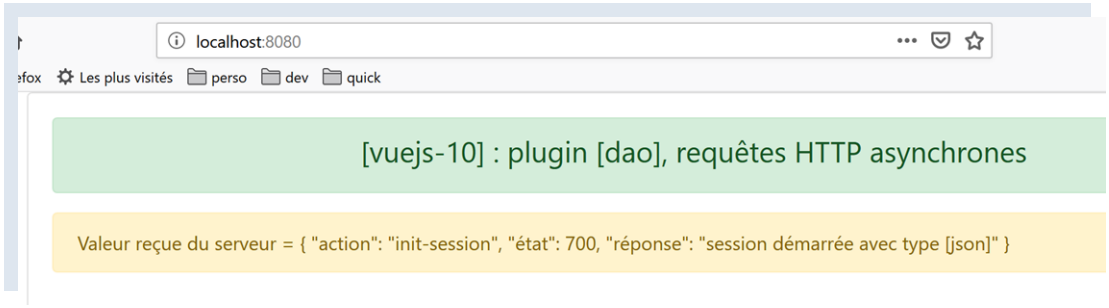
- lignes 49-52 : ce code s'exécute dans tous les cas. On indique au composant parent que la requête HTTP est terminée ;
- lignes 23-28 : la méthode **[doSomethingWithData]** est la méthode chagée d'exploiter la donnée **[data]** envoyée par le serveur. Ligne 25, on enregistre cette donnée et ligne 27 on l'affiche ;

3.12.7 Exécution du projet

Si lorsqu'on lance le projet, le serveur de calcul d'impôt n'est pas lancé alors on obtient le résultat suivant :



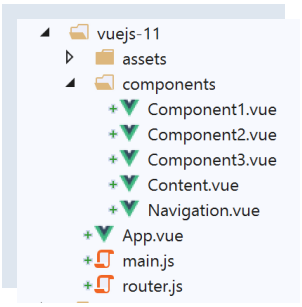
Lançons le serveur **[Laragon]** (cf https://tahe.developpez.com/tutoriels-cours/php7) et rechargeons la page ci-dessus. Le résultat est alors le suivant :



Note : nous utilisons ici la version 14 du serveur de calcul d'impôt définie dans le document |https://tahe.developpez.com/tutoriels-cours/php7|.

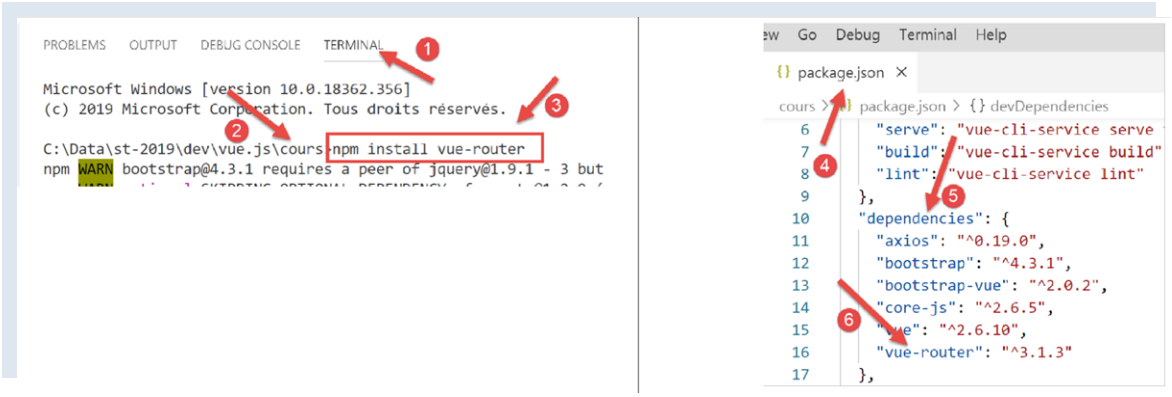
3.13 projet [vuejs-11] : routage et navigation

Le routage est ce qui va permettre à l'utilisateur de naviguer entre les différentes pages de l'application. L'arborescence du projet est la suivante :



3.13.1 Installation des dépendances

Le routage dans **[Vue.js]** nécessite la dépendance **[vue-router]** :



- en **[1-3]**, on installe la dépendance **[vue-router]** ;
- en **[4-6]**, après installation de la dépendance, le fichier **[package.json]** a été modifié ;

3.13.2 Le script de routage [router.js]

Les règles de navigation de l'application sont inscrites dans le fichier **[router.js]** (le nom du script peut être quelconque) :

```
1.  // imports nécessaires au routage
2.  import Vue from "vue";
3.  import VueRouter from "vue-router";
4.
5.  // plugin de routage
6.  Vue.use(VueRouter);
7.
8.  // les composants cibles du routage
9.  import Component1 from "./components/Component1.vue";
10. import Component2 from "./components/Component2.vue";
11. import Component3 from "./components/Component3.vue";
12.
13. // les routes de l'application
14. const routes = [
15.   // home
16.   { path: "/", name: "home", component: Component1 },
17.   // Component1
18.   {
19.     path: "/vue1",
20.     name: "vue1",
21.     component: Component1
22.   },
23.   {
24.     // Component2
25.     path: "/vue2",
26.     name: "vue2",
27.     component: Component2
28.   },
29.   // Component3
30.   {
31.     path: "/vue3",
32.     name: "vue3",
```

```
33.     component: Component3
34.   }
35. ];
36.
37. // le routeur
38. const router = new VueRouter({
39.   routes,
40.   mode: "history"
41. });
42.
43. // export du routeur
44. export default router;
```

Commentaires

- le script **[router.js]** va fixer les règles de routage de notre application ;
- ligne 1-6 : activation du plugin **[vue-router]** nécessaire au routage. Cette activation nécessite l'import de la classe / fonction **[Vue]** (ligne 2) et celui du plugin de routage (ligne 3). Ce plugin a été installé avec la dépendance **[router]** que nous venons d'installer ;
- lignes 8-11 : import des vues cibles du routage ;
- lignes 13-35 : définition du tableau des routes. Chaque élément de ce tableau est un objet avec les propriétés suivantes :
  - **[path]** : l'URL de la vue, celle que nous voulons voir affichée dans le champ **[URL]** du navigateur. On est libre de mettre ce qu'on veut ;
  - **[name]** : le nom de la route. Là également on peut mettre ce qu'on veut ;
  - **[component]** : le composant qui affiche la vue. Là c'est forcément un composant existant ;
    - On aura donc ici quatre routes **[/, /vue1, /vue2, /vue3]**.
- lignes 37-41 : le routeur est une instance de la classe **[VueRouter]** importée ligne 3. Le constructeur de **[VueRouter]** est ici utilisé avec deux paramètres :
  - les routes de l'application ;
  - un mode d'écriture des URL dans le navigateur : le mode par défaut **[hash]** écrit les URL sous la forme **[localhost:8080/#/vue1]** (# est le hash). Le mode **[history]** enlève le # **[localhost:8080/vue1]** ;
- ligne 44 : on exporte le routeur ;

3.13.3 Le script principal [main.js]

Le script principal **[main.js]** est le suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // plugins
6.  import BootstrapVue from 'bootstrap-vue'
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import 'bootstrap/dist/css/bootstrap.css'
11. import 'bootstrap-vue/dist/bootstrap-vue.css'
12.
13. // routeur
14. import monRouteur from './router'
15.
16. // configuration
17. Vue.config.productionTip = false
18.
19. // instanciation projet [App]
20. new Vue({
21.   name: "app",
22.   // vue principale
23.   render: h => h(App),
24.   // routeur
25.   router: monRouteur,
26. }).$mount('#app')
```

Commentaires

- ligne 14 : on importe le routeur exporté par le script **[router.js]** ;
- ligne 25 : ce routeur est passé en paramètre du constructeur de la classe **[Vue]** qui va afficher la vue principale **[App]**, associé à la propriété **[router]** de la vue ;

3.13.4 La vue principale [App]

Le code de la vue principale est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- un message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-11] : routage et navigation</h4>
7.        </b-alert>
8.        <!-- la vue courante du routage -->
9.        <router-view />
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15.   export default {
16.     name: "app"
17.   };
18. </script>
```

Commentaires

- ligne 9 : affiche la vue courante du routage. La balise <**router-view**> n'est reconnue que si la propriété **[router]** de la vue a été initialisée ;

3.13.5 La mise en page des vues

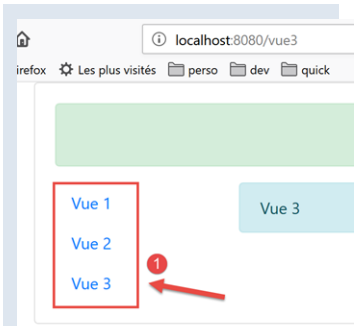
La mise en page des vues est assurée par le composant **[Layout]** suivant :

```
1.  <template>
2.    <!-- ligne -->
3.    <div>
4.      <b-row>
5.        <!-- zone à trois colonnes -->
6.        <b-col cols="2" v-if="left">
7.          <slot name="left" />
8.        </b-col>
9.        <!-- zone à neuf colonnes -->
10.       <b-col cols="10" v-if="right">
11.         <slot name="right" />
12.       </b-col>
13.     </b-row>
14.   </div>
15. </template>
16.
17. <script>
18.   export default {
19.     // paramètres
20.     props: {
21.       left: {
22.         type: Boolean
23.       },
24.       right: {
25.         type: Boolean
26.       }
27.     }
28.   };
29. </script>
```

Nous avons déjà utilisé et expliqué cette mise en page dans le projet **[vuejs-06]** du paragraphe **[vuejs-06]**.

3.13.6 Le composant de navigation

Le composant **[Navigation]** offre un menu de navigation à l'utilisateur :



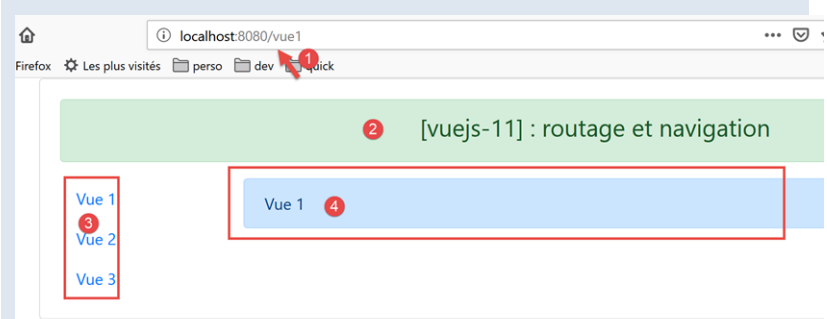
Le composant qui génère le bloc **[1]** est le suivant :

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <b-nav-item to="/vue1" exact exact-active-class="active">Vue 1</b-nav-item>
5.      <b-nav-item to="/vue2" exact exact-active-class="active">Vue 2</b-nav-item>
6.      <b-nav-item to="/vue3" exact exact-active-class="active">Vue 3</b-nav-item>
7.    </b-nav>
8.  </template>
```

- ce code génère le bloc 1 de trois liens permettant la navigation ;
- l'attribut **[to]** des balises <b-nav-item> doit correspondre à l'une des propriétés **[path]** des routes du routeur de l'application ;

3.13.7 Les vues

La vue n° 1 est la suivante :



Les zone **[3-4]** sont générées par le composant **[Component1]** suivant :

```
1.  <!-- vue n° 1 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert show variant="primary" slot="right">Vue 1</b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. import Navigation from "./Navigation";
13. import Layout from "./Layout";
14.
15. export default {
16.   name: "component1",
17.   // composants utilisés
18.   components: {
19.     Layout,
20.     Navigation,
21.   },
22. };
23. </script>
```

Commentaires

- ligne 3 : la vue n° 1 utilise la mise en page du composant **[Layout]** composée de deux slots appelés **[left, right]** ;
- ligne 5 : le menu de navigation est placé dans le slot de gauche. C'est la zone **[3]** qu'on voit ci-dessus ;
- ligne 7 : un message est placé dans le slot de droite. C'est la zone **[4]** qu'on voit ci-dessus ;

Les vues 2 et 3 sont analogues.

Vue n° 2 affichée par le composant **[Component2]** :

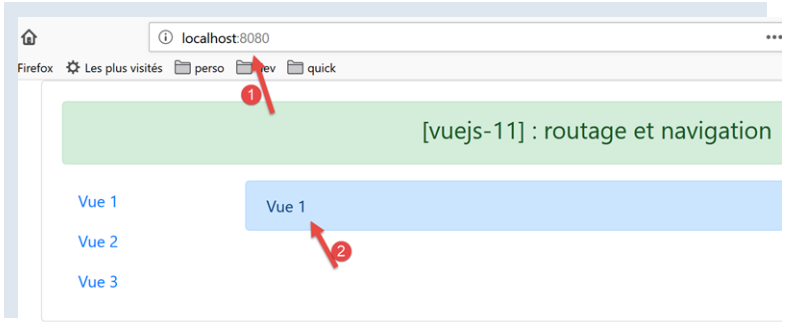
```
1.  <!-- vue n° 2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <b-alert show variant="secondary" slot="right">Vue 2</b-alert>
7.    </Layout>
8.  </template>
9.
10. <script>
11.   import Navigation from './Navigation';
12.   import Layout from './Layout';
13.
14.   export default {
15.     name: "component2",
16.     // composants de la vue
17.     components: {
18.       Layout, Navigation
19.     }
20.   };
21. </script>
```

Vue n° 3 affichée par le composant **[Component3]** :

```
1.  <!-- vue n° 3 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert show variant="info" slot="right">Vue 3</b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12.   import Navigation from "./Navigation";
13.   import Layout from "./Layout";
14.
15.   export default {
16.     name: "component3",
17.     // composants de la vue
18.     components: {
19.       Layout,
20.       Navigation
21.     }
22.   };
23. </script>
```

3.13.8 Exécution du projet

A l'exécution la vue suivante s'affiche :



- en **[1]**, l'URL est **[http://localhost:8080]**. C'est alors la règle de routage suivante qui s'est exécutée :

```
{ path: '/', name: 'home', component: Component1 }
```

C'est donc le composant **[Component1]** qui a été affiché. Il affiche la vue n° 1 **[2]**. Maintenant cliquons sur le lien **[Vue 1]** dont le code est le suivant :

```
<b-nav-item to="/vue1" exact exact-active-class="active">Vue 1</b-nav-item>
```

L'affichage devient le suivant :

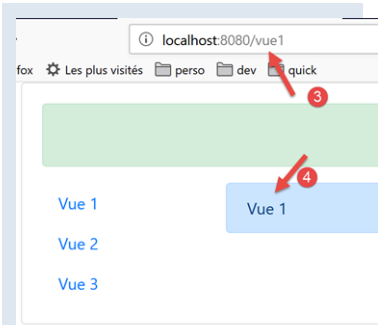
- en **[3]**, la règle suivante de routage s'est exécutée :

```
path: '/vue1', name: 'vue1', component: Component1
```

C'est donc de nouveau le composant **[Component1]** qui a été affiché et donc la vue n° 1 **[4]**. Maintenant cliquons sur le lien **[Vue 2]** dont le code est le suivant :

```
<b-nav-item to="/vue2" exact exact-active-class="active">Vue 2</b-nav-item>
```

La nouvelle vue est alors la suivante :



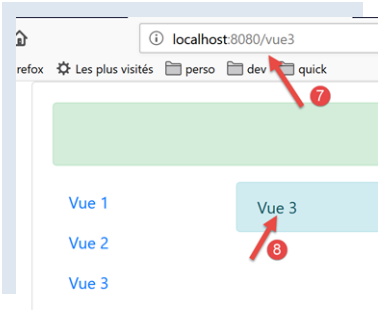
- en **[5]**, la règle suivante de routage s'est exécutée :

```
path: '/vue2', name: 'vue2', component: Component2
```

C'est donc le composant **[Component2]** qui a été affiché et donc la vue n° 2. Si maintenant nous cliquons sur le lien **[Vue 3]**, dont le code est le suivant :

```
<b-nav-item to="/vue3" exact exact-active-class="active">Vue 3</b-nav-item>
```

Nous obtenons la nouvelle vue suivante :



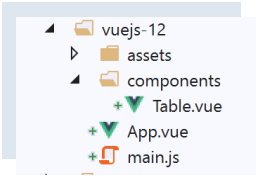
- en **[6]**, la règle de routage suivante s'est exécutée :

```
path: '/vue3', name: 'vue3', component: Component3
```

C'est donc le composant **[Component3]** qui s'est affiché, ç-à-d la vue n° 3 **[8]**.

3.14 projet [vuejs-12] : gestion des tables HTML

L'arborescence du projet **[vuejs-12]** est la suivante :



3.14.1 Le script principal [main.js]

Le script principal redevient ce qu'il était dans les premiers projets :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // plugins
6.  import BootstrapVue from 'bootstrap-vue'
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import 'bootstrap/dist/css/bootstrap.css'
11. import 'bootstrap-vue/dist/bootstrap-vue.css'
12.
13. // configuration
14. Vue.config.productionTip = false
15.
16. // instanciation projet [App]
17. new Vue({
18.   name: "app",
19.   render: h => h(App),
20. }).$mount('#app')
```

3.14.2 La vue principale [App]

Le code de la vue principale **[App]** est le suivant :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-12] : gestion des tables</h4>
7.        </b-alert>
8.        <!-- composant Table -->
9.        <Table @supprimerLigne="supprimerLigne" :lignes="lignes" @rechargerListe="rechargerListe" />
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. import Table from "./components/Table";
16. export default {
17.   // nom
18.   name: "app",
19.   // composants utilisés
20.   components: {
21.     Table,
22.   },
23.   // état interne
24.   data() {
25.     return {
26.       // liste des simulations
27.       lignes: [
28.         { id: 3, marié: "oui", enfants: 2, salaire: 35000, impôt: 1200 },
29.         { id: 5, marié: "non", enfants: 2, salaire: 35000, impôt: 1900 },
30.         { id: 7, marié: "non", enfants: 0, salaire: 30000, impôt: 2000 },
31.       ],
32.     };
33.   },
34.
35.   // méthodes
36.   methods: {
37.     // suppression d'une ligne
38.     supprimerLigne(index) {
39.       // eslint-disable-next-line
40.       console.log("App supprimerLigne", index);
41.       // suppression ligne à la position [index]
42.       this.lignes.splice(index, 1);
43.     },
44.     // rechargement de la table des lignes
45.     rechargerListe() {
46.       // eslint-disable-next-line
47.       console.log("App rechargerListe");
48.       // on régénère la liste des lignes
49.       this.lignes = [
50.         { id: 3, marié: "oui", enfants: 2, salaire: 35000, impôt: 1200 },
51.         { id: 5, marié: "non", enfants: 2, salaire: 35000, impôt: 1900 },
52.         { id: 7, marié: "non", enfants: 0, salaire: 30000, impôt: 2000 },
53.       ];
54.     },
55.   },
56. };
57. </script>
```

Commentaires

- la vue principale affiche un composant **[Table]** (lignes 9, 15, 21) ;
- ligne 9 : le composant **[Table]** admet le paramètre **[:lignes]** qui représente les lignes à afficher dans une table HTML. Ces lignes sont définies par les lignes 27-31 du code ;
- ligne 9 : le composant **[Table]**est susceptible d'émettre deux événements :
  - **[supprimerLigne]** : pour supprimer une ligne dont on donne l'index (ligne 38) ;
  - **[rechargerListe]** : pour régénérer la liste des lignes 27-31. En effet, nous allons voir que l'utilisateur peut supprimer certaines des lignes affichées ;
- lignes 38-43 : la méthode chargée de gérer l'événement **[supprimerLigne]**. Elle reçoit en paramètre l'index de la ligne à supprimer ;

- lignes 45-54 : la méthode chargée de gérer l'événement **[rechargerListe]**. En modifiant l'attribut **[lignes]** de la ligne 27, on provoque la mise à jour du composant **[Table]** de la ligne 9, puisqu'il a un paramètre **[:lignes="lignes"]** ;

3.14.3 Le composant [Table]

Le composant **[Table]** est le suivant :

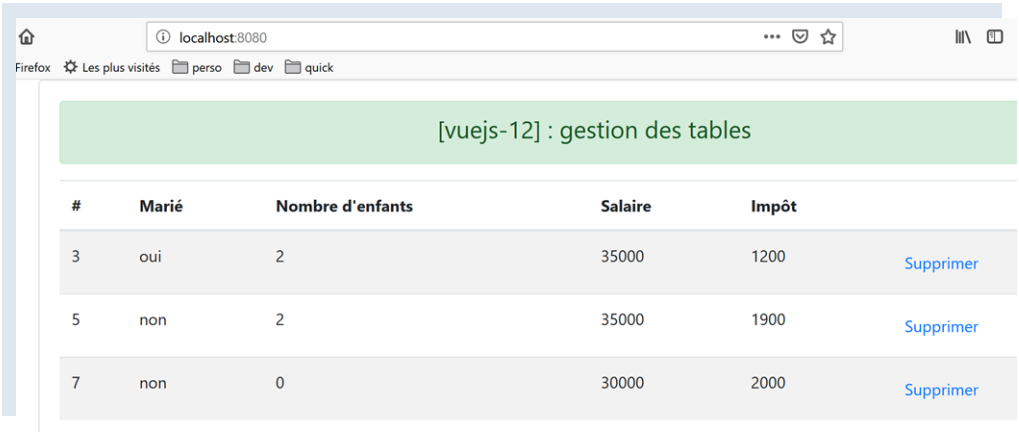
```
1.  <template>
2.    <div>
3.      <!-- liste vide -->
4.      <template v-if="lignes.length==0">
5.        <b-alert show variant="warning">
6.          <h4>Votre liste de simulations est vide</h4>
7.        </b-alert>
8.        <!-- bouton de rechargement-->
9.        <b-button variant="primary" @click="rechargerListe">Recharger la liste</b-button>
10.     </template>
11.     <!-- liste non vide-->
12.     <template v-if="lignes.length!=0">
13.       <b-alert show variant="primary" v-if="lignes.length==0">
14.         <h4>Liste de vos simulations</h4>
15.       </b-alert>
16.       <!-- tableau des simulations -->
17.       <b-table striped hover responsive :items="lignes" :fields="fields">
18.         <template v-slot:cell(action)="row">
19.           <b-button variant="link" @click="supprimerLigne(row.index)">Supprimer</b-button>
20.         </template>
21.       </b-table>
22.     </template>
23.   </div>
24. </template>
25.
26. <script>
27. export default {
28.   // propriétés
29.   props: {
30.     lignes: {
31.       type: Array,
32.     },
33.   },
34.   // état interne
35.   data() {
36.     return {
37.       fields: [
38.         { label: "#", key: "id" },
39.         { label: "Marié", key: "marié" },
40.         { label: "Nombre d'enfants", key: "enfants" },
41.         { label: "Salaire", key: "salaire" },
42.         { label: "Impôt", key: "impôt" },
43.         { label: "", key: "action" },
44.       ],
45.     };
46.   },
47.   // méthodes
48.   methods: {
49.     // suppression d'une ligne
50.     supprimerLigne(index) {
51.       // eslint-disable-next-line
52.       console.log("Table supprimerLigne", index);
53.       // on passe l'information au composant parent
54.       this.$emit("supprimerLigne", index);
55.     },
56.     // rechargement de la liste affichée
57.     rechargerListe() {
58.       // eslint-disable-next-line
59.       console.log("Table rechargerListe");
60.       // on émet un événement vers le composant parent
61.       this.$emit("rechargerListe");
62.     },
63.   },
64. };
65. </script>
```

Commentaires

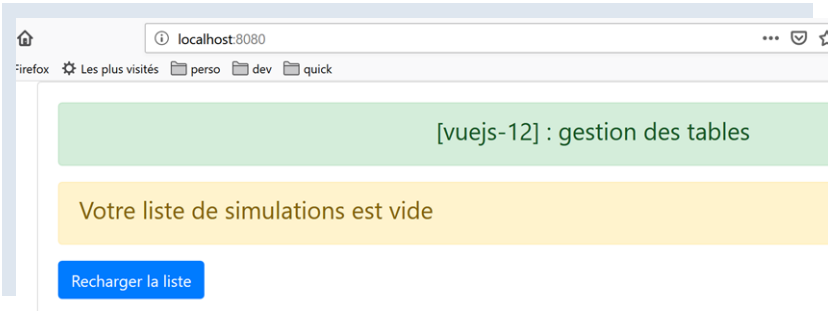
Ce composant a deux états :

1 il affiche une liste dans un tableau HTML ;
2 ou bien un message indiquant que la liste à afficher est vide ;

Le 1er état est affiché si la condition **[lignes.length!=0]** est vérifiée (ligne 12) :



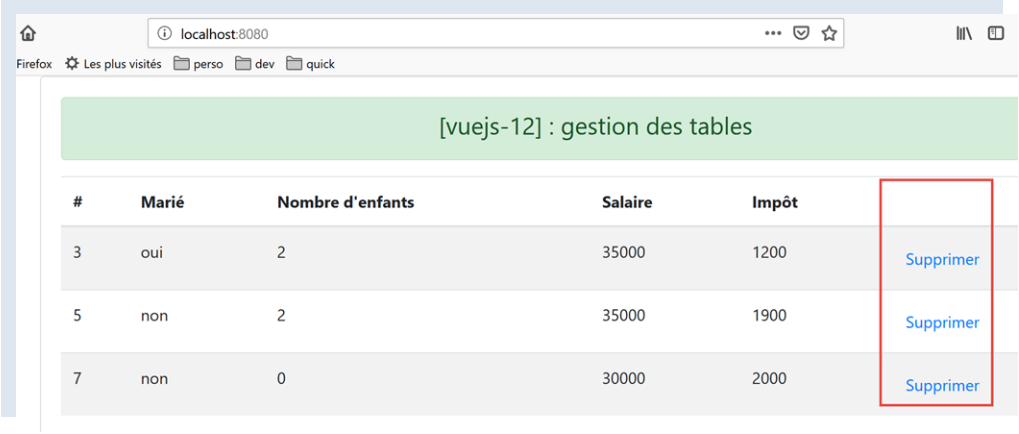
Le second est affiché si la condition **[lignes.length==0]** est vérifiée (ligne 4).



- **[lignes]** est un paramètre d'entrée du composant (lignes 29-33) ;
- lignes 4-10 : au lieu d'introduire un bloc de code avec une balise , on a ici utilisé une balise <template>. la différence entre les deux est que la balise <template> n'est pas insérée dans le code HTML généré ;
- ligne 9 : lorsque la liste affichée par la table HTML est vide, un bouton propose de la régénérer. Lorsqu'on clique sur ce bouton, la méthode **[rechargerListe]** des lignes 57-62 est exécutée. Celle-ci se contente d'émettre vers le composant parent l'événement **[rechargerListe]** qui demande au parent de régénérer la liste affichée par la table HTML ;
- lignes 12-22 : le code affiché lorsque la liste à afficher n'est pas vide ;
- ligne 17 : la balise <b-table> est la balise qui génère une table HTML. Les attributs utilisés ici sont les suivants :
  - **[striped]** : la couleur de fond des lignes alterne. Il y a une couleur pour les lignes paires et une autre pour les lignes impaires. Cela améliore la visibilité ;
  - **[hover]** : la ligne sur laquelle on passe la souris change de couleur ;
  - **[responsive]** : la taille de la table s'adapte à l'écran qui l'affiche ;
  - **[:items]** : le tableau des éléments à afficher. Ici le tableau **[lignes]** passé en paramètre au composant (lignes 30-32) ;
  - **[:fields]** : un tableau définissant la mise en page de la table HTML (lignes 37-44) ;
    - chaque élément du tableau **[fields]** définit une colonne de la table HTML ;
    - **[label]** : désigne le titre de la colonne ;
    - **[key]** : désigne le contenu de la colonne ;
- ligne 38 : définit la colonne 0 de la table HTML :
  - **[#]** : est le titre de la colonne ;
  - **[id]** : est son contenu. C'est le champ **[id]** de la ligne affichée qui remplira la colonne 0 ;
- ligne 39 : définit la colonne 1 de la table HTML :
  - **[Marié]** : est le titre de la colonne ;
  - **[marié]** : est son contenu. C'est le champ **[marié]** de la ligne affichée qui remplira la colonne 0 ;
- ligne 43 : définit la dernière colonne de la table HTML :
  - la colonne n'a pas de titre ;
  - son contenu est défini par un champ **[action]**, un champ qui n'existe pas dans les lignes affichées. Cette clé est référencée ligne 18 du **[template]**. La clé est donc utilisée ici uniquement pour identifier une colonne ;
- lignes 18-20 : ce code sert à définir la dernière colonne de la table HTML, celle de clé **[action]** :

```
<template v-slot:cell(action)="row">
```

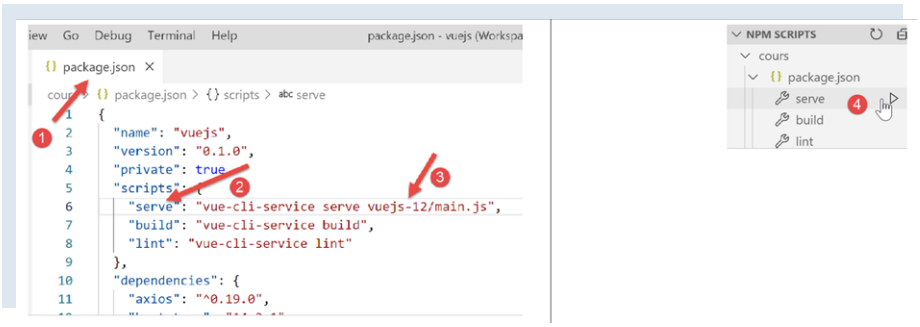
La syntaxe **[v-slot:cell(action)]** désigne la colonne de clé **[action]**. Cette syntaxe permet de définir une colonne lorsque la syntaxe du tableau **[fields]** n'est pas suffisante pour la décrire. Ici nous voulons que la dernière colonne contienne un lien permettant de supprimer une ligne de la table HTML :



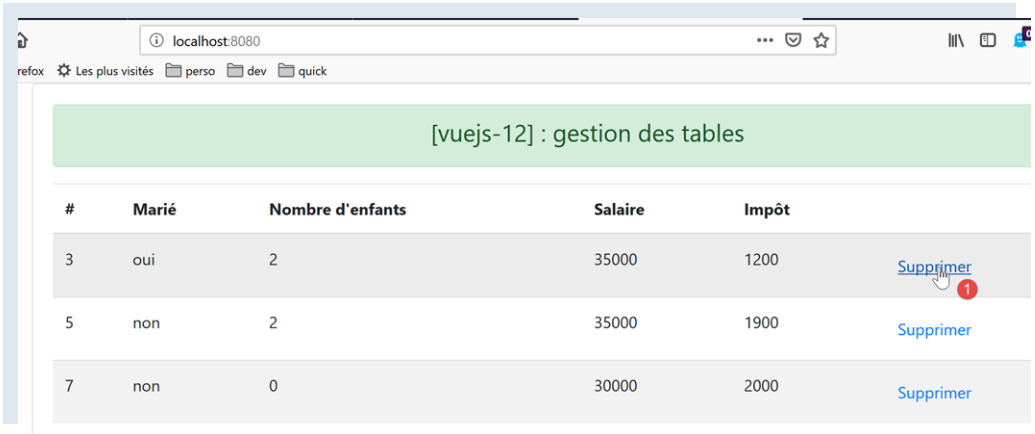
Dans la syntaxe **[<template v-slot:cell(action)="row">]**, le nom **[row]** désigne la ligne de la table. On peut utiliser le nom que l'on veut. On aurait pu écrire **[<template v-slot:cell(action)="ligne">]** ;

- ligne 19 : un bouton <b-button> affiché comme un lien **[variant='link']**. Un clic sur ce lien provoque l'exécution de la méthode **[supprimerLigne(row.index)]**. Ici **[row]** est le nom donné à la ligne de la table HTML, ligne 18 du code ;
- lignes 50-55 : la méthode **[supprimerLigne]** ;
- ligne 54 : on émet l'événement **[supprimerLigne]** vers le composant parent accompagné du n° de la ligne à supprimer ;
- lignes 57-62 : la méthode **[rechargerListe]** ;
- ligne 61 : on émet l'événement **[rechargerListe]** vers le composant parent ;

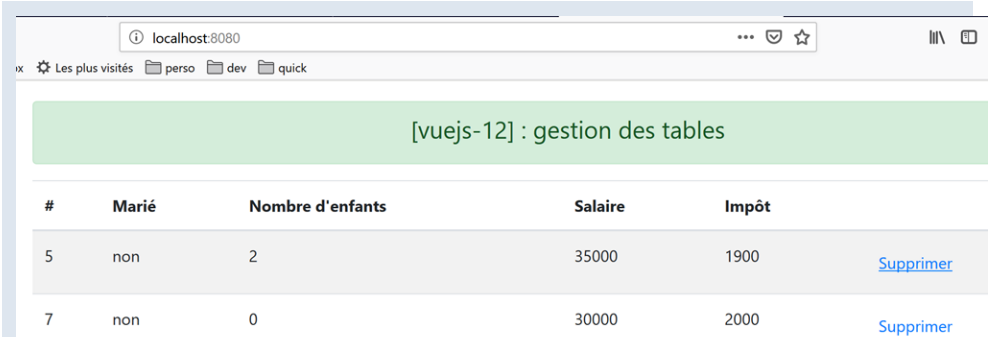
3.14.4 Exécution du projet



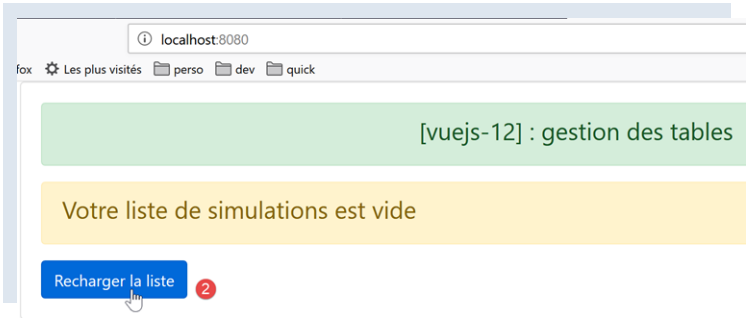
La 1ère vue affichée est la suivante :



Après avoir supprimé la ligne 1 **[1]**, la vue devient la suivante :



Après avoir supprimé toutes les lignes :

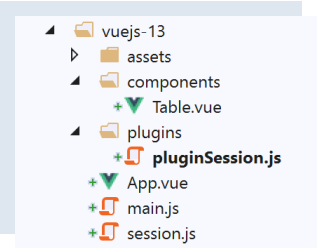


Après avoir cliqué sur le bouton **[2]**, on obtient de nouveau la liste :

3.15 projet [vuejs-13] : mise à jour d'un composant, utilisation d'une session

Le projet **[vuejs-13]** reprend le projet **[vuejs-12]** en amenant la modification suivante : le tableau affiché par la table HTML est défini dans un objet **[session]** connu de tous les composants. C'est donc une façon de partager de l'information entre composants. Ce concept est directement inspiré de la session web. Nous utilisons la méthode du plugin pour rendre disponible cet objet partagé dans un attribut **[Vue.$session]**.

L'arborescence du projet est la suivante :



3.15.1 L'objet `[session]`

L'objet **`[session]`** partagé par tous les composants est défini dans le script **`[./session.js]`** :

```
1.  const session = {
2.    // liste des simulations
3.    get lignes() {
4.      return this._lignes;
5.    },
6.    // génération de la liste des simulations
7.    generateLignes() {
8.      this._lignes =
9.        [
10.         { id: 3, marié: "oui", enfants: 2, salaire: 35000, impôt: 1200 },
11.         { id: 5, marié: "non", enfants: 2, salaire: 35000, impôt: 1900 },
12.         { id: 7, marié: "non", enfants: 0, salaire: 30000, impôt: 2000 }
13.       ]
14.   },
15.   // suppression ligne n° index
16.   deleteLigne(index) {
17.     this._lignes.splice(index, 1);
18.   }
19. }
20. // export de l'objet [session]
21. export default session;
```

3.15.2 Le plugin `[./plugins/pluginSession]`

Le script **`[pluginSession]`** est le suivant :

```
1.  export default {
2.    install(Vue, session) {
3.      // ajoute une propriété [$session] à la classe vue
4.      Object.defineProperty(Vue.prototype, '$session', {
5.        // lorsque Vue.$session est référencé, on rend le 2ième paramètre [session] de la méthode [install]
6.        get: () => session,
7.      })
8.    }
9.  }
```

- ligne 4 : l'objet partagé **`[session]`** sera disponible dans la propriété **`[$session]`** de tous les composants ;

3.15.3 Le script principal `[main.js]`

Le script principal **`[main.js]`** est le suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // plugins
6.  import BootstrapVue from 'bootstrap-vue'
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import 'bootstrap/dist/css/bootstrap.css'
11. import 'bootstrap-vue/dist/bootstrap-vue.css'
12.
13. // session
14. import session from './session';
15. import pluginSession from './plugins/pluginSession'
16. Vue.use(pluginSession, session)
```

```
17.
18. // configuration
19. Vue.config.productionTip = false
20.
21. // instanciation projet [App]
22. new Vue({
23.   name: "app",
24.   render: h => h(App),
25. }).$mount('#app')
```

- lignes 14-16 : le plugin **[pluginSession]** est intégré au framework **[Vue.js]** ;
- après la ligne 16, l'attribut **[$session]** est disponible pour tous les composants ;

3.15.4 La vue principale [App]

La vue **[App]** est désormais la suivante :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-13] : mise à jour d'un composant, utilisation d'un objet partagé entre composants</h4>
7.        </b-alert>
8.        <!-- table HTML -->
9.        <Table @updateTable="updateTable" :key=" versionTable" />
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. import Table from "./components/Table";
16. export default {
17.   // nom
18.   name: "app",
19.   // composants
20.   components: {
21.     Table
22.   },
23.   // état interne
24.   data() {
25.     return {
26.       // version table
27.       versionTable: 1
28.     };
29.   },
30.   // méthodes
31.   methods: {
32.     // mise à jour du composant [Table]
33.     updateTable() {
34.       // eslint-disable-next-line
35.       console.log("App updateTable");
36.       // incrément version table
37.       this.versionTable++;
38.     }
39.   },
40.   // cycle de vie
41.   created() {
42.     // génération du tableau des simulations
43.     this.$session.generateLignes();
44.   }
45. };
46. </script>
```

Commentaires

- la vue **[App]** ne gère plus désormais le tableau affiché par le composant **[Table]** de la ligne 9 ;
- ligne 9 : le composant **[Table]** émet l'événement **[updateTable]** qui demande à ce que le composant **[Table]** soit régénéré. Une façon de faire cela est d'utiliser l'attribut **[:key]**. On donne à cet attribut une valeur modifiable. A chaque fois qu'elle est modifiée, le composant **[Table]** est régénéré ;
- ligne 9 : la valeur de l'attribut **[:key]** est l'attribut **[versionTable]** de la ligne 27. La méthode **[updateTable]** (lignes 33-38) est chargée de régénérer le composant **[Table]** de la ligne 9. Pour cela, la méthode incrémente la valeur de l'attribut **[:key]** du composant **[Table]**, ligne 37. Le composant **[Table]** est alors automatiquement régénéré ;

3.15.5 Le composant `[Table]`

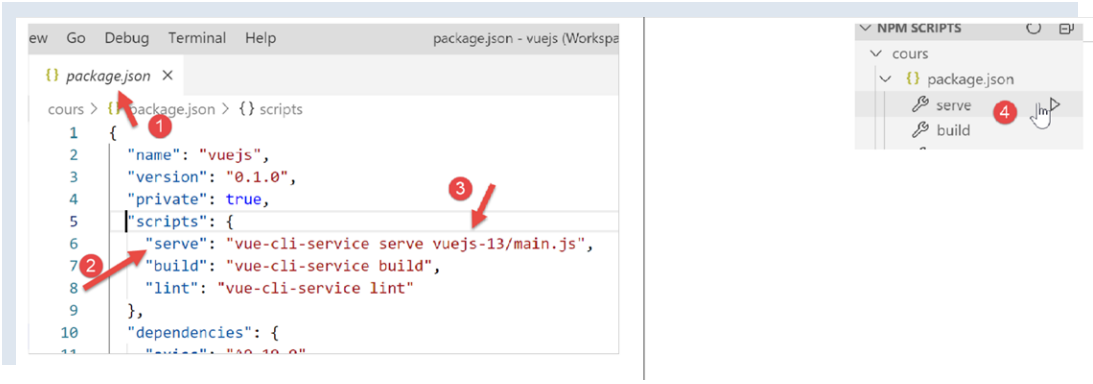
Le composant **`[Table]`** évolue de la façon suivante :

```
<template>
  <div>
    <!-- liste vide -->
    <template v-if="lignes.length==0">
      <b-alert show variant="warning">
        <h4>Votre liste de simulations est vide</h4>
      </b-alert>
      <!-- bouton de rechargement-->
      <b-button variant="primary" @click="rechargerListe">Recharger la liste</b-button>
    </template>
    <!-- liste non vide-->
    <template v-if="lignes.length!=0">
      <b-alert show variant="primary">
        <h4>Liste de vos simulations</h4>
      </b-alert>
      <!-- tableau des simulations -->
      <b-table striped hover responsive :items="lignes" :fields="fields">
        <template v-slot:cell(action)="row">
          <b-button variant="link" @click="supprimerLigne(row.index)">Supprimer</b-button>
        </template>
      </b-table>
    </template>
  </div>
</template>

<script>
export default {
  // état calculé
  computed: {
    lignes() {
      return this.$session.lignes;
    }
  },
  // état interne
  data() {
    return {
      fields: [
        { label: "#", key: "id" },
        { label: "Marié", key: "marié" },
        { label: "Nombre d'enfants", key: "enfants" },
        { label: "Salaire", key: "salaire" },
        { label: "Impôt", key: "impôt" },
        { label: "", key: "action" }
      ]
    };
  },
  // méthodes
  methods: {
    supprimerLigne(index) {
      // eslint-disable-next-line
      console.log("Table supprimerLigne", index);
      // on supprime la ligne
      this.$session.deleteLigne(index);
      // on demande au composant parent de mettre à jour la vue
      this.$emit("updateTable");
    },
    // rechargement de la liste affichée
    rechargerListe() {
      // eslint-disable-next-line
      console.log("Table rechargerListe");
      // on régènère la liste des simulations
      this.$session.generateLignes();
      // on demande au composant parent de mettre à jour la vue
      this.$emit("updateTable");
    }
  }
};
</script>
```

Commentaires :

- l'attribut **[lignes]** (lignes 4, 12, 17) n'est plus un paramètre fixé par le composant parent mais un attribut calculé du composant **[Table]** (lignes 30-32). **[lignes]** est alors le tableau **[$session.lignes]** (ligne 31) ;
- lignes 49-56 : la méthode **[supprimerLigne]** fait supprimer une ligne du tableau **[$session.lignes]**. Cette suppression ne change pas, par défaut, l'affichage de la table HTML. En effet, les éléments de **[$session] ne sont pas réactifs** : leur modification n'est pas répercutée sur les composants qui les utilisent. Pour cette raison, le composant **[Table]** demande à son parent de le régénérer au moyen de l'événement **[updateTable]** (ligne 55). On a vu que le composant parent allait alors incrémenter l'attribut **[:key]** du composant **[Table]** pour forcer sa régénération ;
- lignes 58-65 : la méthode **[rechargerListe]** demande à l'objet **[$session]** de régénérer le tableau **[$session.lignes]**. Pour la même raison que précédemment, cette modification de **[$session.liste]** ne change pas, par défaut, l'affichage de la table HTML. Pour cette raison, le composant **[Table]** demande à son parent de le régénérer au moyen de l'événement **[updateTable]** (ligne 64).

3.15.6 Exécution du projet

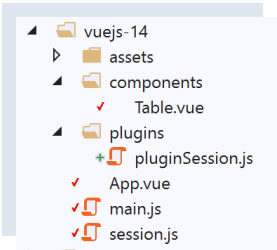


On obtient les mêmes résultats que dans le projet **[vuejs-12]**.

3.16 projet [vuejs-14] : rendre la session réactive

On a vu que l'objet **[session]** utilisé dans le projet précédent avait des propriétés **non réactives** : si on les modifie, les vues utilisant ces propriétés ne sont pas mise à jour. Il est possible d'avoir un objet **[session]** réactif si on le stocke dans les données réactives des vues. C'est ce que montre le projet **[vuejs-14]**.

L'arborescence du projet est la suivante :



3.16.1 L'objet [session]

L'objet **[session]** partagé par tous les composants ne change pas.

3.16.2 Le plugin [./plugins/pluginSession]

Le script **[pluginSession]** ne change pas. L'objet partagé **[session]** est disponible dans la propriété **[$session]** de tous les composants.

3.16.3 Le script principal [main.js]

Le script principal **[main.js]** ne change pas.

3.16.4 La vue principale [App]

La vue **[App]** est désormais la suivante :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-14] : utilisation d'un objet partagé entre composants</h4>
7.        </b-alert>
8.        <!-- table HTML -->
9.        <Table/>
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. import Table from "./components/Table";
16. export default {
17.   // nom
18.   name: "app",
19.   // composants
20.   components: {
21.     Table
22.   },
23.   // cycle de vie
24.   created() {
25.     // génération du tableau des simulations
26.     this.$session.generateLignes();
27.   }
28. };
29. </script>
```

Commentaires

- ligne 9 : le composant **[Table]** n'émet plus l'événement **[updateTable]** qui demande à ce que le composant **[Table]** soit régénéré. Du coup, la méthode **[updateTable]** a disparu ;

3.16.5 Le composant [Table]

Le composant **[Table]** évolue de la façon suivante :

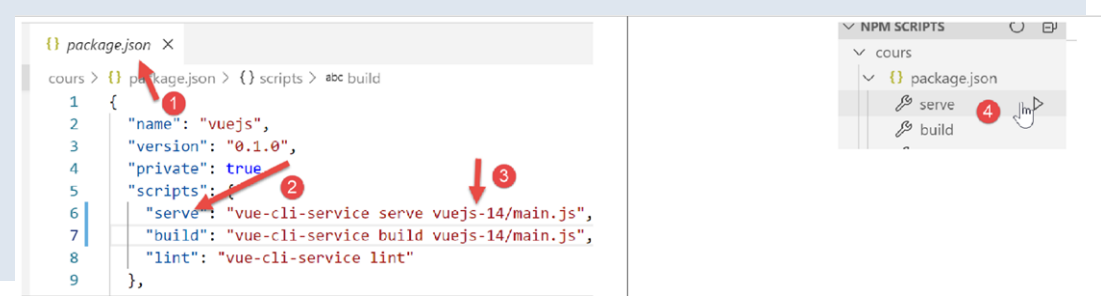
```
1.  <template>
2.    <div>
3.      <!-- liste vide -->
4.      <template v-if="lignes.length==0">
5.        <b-alert show variant="warning">
6.          <h4>Votre liste de simulations est vide</h4>
7.        </b-alert>
8.        <!-- bouton de rechargement-->
9.        <b-button variant="primary" @click="rechargerListe">Recharger la liste</b-button>
10.     </template>
11.     <!-- liste non vide-->
12.     <template v-if="lignes.length!=0">
13.       <b-alert show variant="primary" v-if="lignes.length==0">
14.         <h4>Liste de vos simulations</h4>
15.       </b-alert>
16.       <!-- tableau des simulations -->
17.       <b-table striped hover responsive :items="lignes" :fields="fields">
18.         <template v-slot:cell(action)="row">
19.           <b-button variant="link" @click="supprimerLigne(row.index)">Supprimer</b-button>
20.         </template>
21.       </b-table>
22.     </template>
23.   </div>
24. </template>
25.
26. <script>
27. export default {
28.   // état calculé
29.   computed: {
30.     lignes() {
31.       return this.session.lignes;
32.     }
33.   },
34.   // état interne
35.   data() {
```

```
36.     return {
37.       fields: [
38.         { label: "#", key: "id" },
39.         { label: "Marié", key: "marié" },
40.         { label: "Nombre d'enfants", key: "enfants" },
41.         { label: "Salaire", key: "salaire" },
42.         { label: "Impôt", key: "impôt" },
43.         { label: "", key: "action" }
44.       ],
45.       session : {}
46.     };
47.   },
48.   // cycle de vie
49.   created(){
50.     this.session=this.$session
51.   },
52.   // méthodes
53.   methods: {
54.     supprimerLigne(index) {
55.       // eslint-disable-next-line
56.       console.log("Table supprimerLigne", index);
57.       // on supprime la ligne
58.       this.session.deleteLigne(index);
59.     },
60.     // rechargement de la liste affichée
61.     rechargerListe() {
62.       // eslint-disable-next-line
63.       console.log("Table rechargerListe");
64.       // on régènère la liste des simulations
65.       this.session.generateLignes();
66.     }
67.   }
68. };
69. </script>
```

Commentaires :

- la nouveauté est lignes 49-51 : lorsque la vue est créée, la session **[this.$session]** est stockée dans la propriété **[session]** de la ligne 45. Placée ici, la propriété **[session]** est réactive ;
- lignes 58 et 65 : au lieu d'utiliser **[this.$session]** pour ajouter / supprimer une ligne de la table, on utilise la propriété réactive **[this.session]** ;

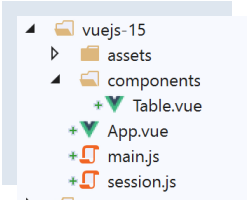
3.16.6 Exécution du projet



On obtient les mêmes résultats que dans le projet **[vuejs-12]**.

3.17 projet [vuejs-15] : utilisation du plugin [Vuex]

Le projet **[vuejs-15]** reprend le projet **[vuejs-14]** en utilisant un objet **[session]** réactif généré par **[Vuex]**. L'aborescence du projet est la suivante :



3.17.1 Installation de la dépendance [vuex]

3.17.2 Le script [./session.js]

L'objet **[session]** devient le suivant :

```
1.  // plugin Vuex
2.  import Vue from 'vue'
3.  import Vuex from 'vuex'
4.  Vue.use(Vuex);
5.
6.  // la session est un store Vuex
7.  const session = new Vuex.Store({
8.    state: {
9.      // le tableau des simulations
10.     lignes: []
11.   },
12.   mutations: {
13.     // génération de la liste des simulations
14.     generateLignes(state) {
15.       // eslint-disable-next-line no-console
16.       console.log("mutation generateLignes");
17.       // on initialise [state.lignes]
18.       state.lignes =
19.         [
20.           { id: 3, marié: "oui", enfants: 2, salaire: 35000, impôt: 1200 },
21.           { id: 5, marié: "non", enfants: 2, salaire: 35000, impôt: 1900 },
22.           { id: 7, marié: "non", enfants: 0, salaire: 30000, impôt: 2000 }
23.         ];
24.       // eslint-disable-next-line no-console
25.       console.log(state.lignes);
26.     },
27.     // suppression ligne n° index
28.     deleteLigne(state, index) {
29.       // eslint-disable-next-line no-console
30.       console.log("mutation deleteLigne");
31.       // on supprime la ligne n° [index]
32.       state.lignes.splice(index, 1);
33.     }
34.   }
35. });
36. // export de l'objet [session]
37. export default session;
```

Commentaires

- lignes 2-4 : on intègre le plugin **[Vuex]** au framework **[Vue]** ;
- ligne 7 : la session devient un objet de type **[Vuex.Store]** ;
- lignes 8-11 : la propriété **[state]** contient l'état partagé de l'application **[Vue]**. Cette propriété sera accessible à tous les composants de l'application. Ici nous partageons le tableau des simulations **[lignes]** (ligne 10) ;
- lignes 12-35 : la propriété **[mutations]** rassemble les méthodes qui modifie le contenu de l'objet **[state]** ;
- lignes 14-26 : la propriété **[generateLignes]** est une fonction générant une valeur initiale pour la propriété **[state.lignes]**. Elle admet ici **[state]** comme paramètre. Lignes 18-23 : la propriété **[state.lignes]** est initialisée ;
- lignes 28-35 : la propriété **[deleteLigne]** est une fonction supprimant une ligne du tableau **[state.lignes]**. Elle a pour paramètres :
  - **[state]** qui représente l'objet des lignes 8-11 ;
  - **[index]** qui est le n° de la ligne à supprimer ;
- les fonctions de la propriété **[mutations]** admettent toujours comme 1er paramètre, un objet représentant la propriété **[state]** de la ligne 8. Les paramètres suivants sont fournis par le code appelant la mutation ;
- ligne 37 : l'objet **[session]** est exporté.

Contrairement au projet précédent **[vuejs-13]** nous n'aurons pas ici de plugin pour rendre la session accessible aux composants dans un attribut **[Vue.$session]**.

3.17.3 Le script principal [./main.js]

Le script principal évolue de la façon suivante :

```
1.  // imports
2.  import Vue from "vue";
3.  import App from "./App.vue";
4.
5.  // plugins
6.  import BootstrapVue from "bootstrap-vue";
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import "bootstrap/dist/css/bootstrap.css";
11. import "bootstrap-vue/dist/bootstrap-vue.css";
12.
13. // session
14. import session from "./session";
15.
16. // configuration
17. Vue.config.productionTip = false;
18.
19. // instanciation projet [App]
20. new Vue({
21.   name: "app",
22.   // utilisation store de Vuex
23.   store: session,
24.   render: h => h(App)
25. }).$mount("#app");
```

Commentaires

- ligne 14 : la session est importée ;
- ligne 23 : elle est passée à la vue principale dans un attribut nommé **[store]** (c'est imposé). Grâce au plugin **[Vuex]**, cet attribut devient alors disponible à tous les composants dans un attribut **[Vue.$store]**. On est donc dans une configuration très proche de celle du projet précédent : là où dans un composant on accédait à la session via la notation **[this.$session]**, on y accèdera maintenant via la notation **[this.$store]** ;

3.17.4 La vue principale [App]

La vue principale **[App]** évolue comme suit :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[vuejs-15] : utilisation du plugin [Vuex]</h4>
7.        </b-alert>
8.        <!-- table HTML -->
```

```
9.        <Table />
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. import Table from "./components/Table";
16. export default {
17.   // nom
18.   name: "app",
19.   // composants
20.   components: {
21.     Table
22.   },
23.   // cycle de vie
24.   created() {
25.     // génération du tableau des simulations
26.     this.$store.commit("generateLignes");
27.   }
28. };
29. </script>
```

Commentaires

- ligne 9 : la vue **[App]** utilise le composant **[Table]** mais ne reçoit plus d'événements de sa part, ceci grâce au fait que le store **[Vuex]** est **réactif** ;
- lignes 24-27 : la méthode **[created]** est exécutée juste après la création du composant **[App]**. Dans celle-ci, on exécute la mutation nommée **[generateLignes]** qui génère une valeur initiale pour le tableau des simulations. On notera la syntaxe particulière de l'instruction. On rappelle que la notation **[this.$store]** fait référence à la propriété **[store]** de la vue instanciée dans **[main.js]** :

```
1.  // instanciation vue [App]
2.  new Vue({
3.    name: "app",
4.    // utilisation store de Vuex
5.    store: session,
6.    render: h => h(App),
7.  }).$mount('#app')
```

La notation **[this.$store]** désigne donc l'objet **[session]**. On écrit ensuite **[this.$store.commit("generateLignes")]** pour exécuter la mutation s'appelant **[generateLignes]**. Cette mutation est une fonction ;

3.17.5 Le composant [Table]

Le composant **[Table]** évolue de la façon suivante :

```
1.  <template>
2.    <div>
3.      <!-- liste vide -->
4.      <template v-if="lignes.length==0">
5.        <b-alert show variant="warning">
6.          <h4>Votre liste de simulations est vide</h4>
7.        </b-alert>
8.        <!-- bouton de rechargement-->
9.        <b-button variant="primary" @click="rechargerListe">Recharger la liste</b-button>
10.     </template>
11.     <!-- liste non vide-->
12.     <template v-if="lignes.length!=0">
13.       <b-alert show variant="primary">
14.         <h4>Liste de vos simulations</h4>
15.       </b-alert>
16.       <!-- tableau des simulations -->
17.       <b-table striped hover responsive :items="lignes" :fields="fields">
18.         <template v-slot:cell(action)="row">
19.           <b-button variant="link" @click="supprimerLigne(row.index)">Supprimer</b-button>
20.         </template>
21.       </b-table>
22.     </template>
23.   </div>
24. </template>
25.
26. <script>
27. export default {
28.   // état calculé
```

```
29.   computed: {
30.     lignes() {
31.       return this.$store.state.lignes;
32.     }
33.   },
34.   // état interne
35.   data() {
36.     return {
37.       fields: [
38.         { label: "#", key: "id" },
39.         { label: "Marié", key: "marié" },
40.         { label: "Nombre d'enfants", key: "enfants" },
41.         { label: "Salaire", key: "salaire" },
42.         { label: "Impôt", key: "impôt" },
43.         { label: "", key: "action" }
44.       ]
45.     };
46.   },
47.   // méthodes
48.   methods: {
49.     supprimerLigne(index) {
50.       // eslint-disable-next-line
51.       console.log("Table supprimerLigne", index);
52.       // on supprime la ligne
53.       this.$store.commit("deleteLigne", index);
54.     },
55.     // rechargement de la liste affichée
56.     rechargerListe() {
57.       // eslint-disable-next-line
58.       console.log("Table rechargerListe");
59.       // on régènère la liste des simulations
60.       this.$store.commit("generateLignes");
61.     }
62.   }
63. };
64. </script>
```

Commentaires

- le **[template]** des lignes 1-24 ne change pas ;
- lignes 30-32 : la propriété calculée **[lignes]** utilise désormais le **[store]** de **[Vuex]** ;
- lignes 49-54 : pour supprimer une ligne de la table HTML, on utilise la mutation **[deleteLigne]** du **[store]** de **[Vuex]**. On passe en paramètre le n° **[index]** de la ligne à supprimer (ligne 53) ;
- lignes 56-61 : pour recharger la table HTML avec une nouvelle liste, on utilise la mutation **[generateLignes]** du **[store]** de **[Vuex]** ;

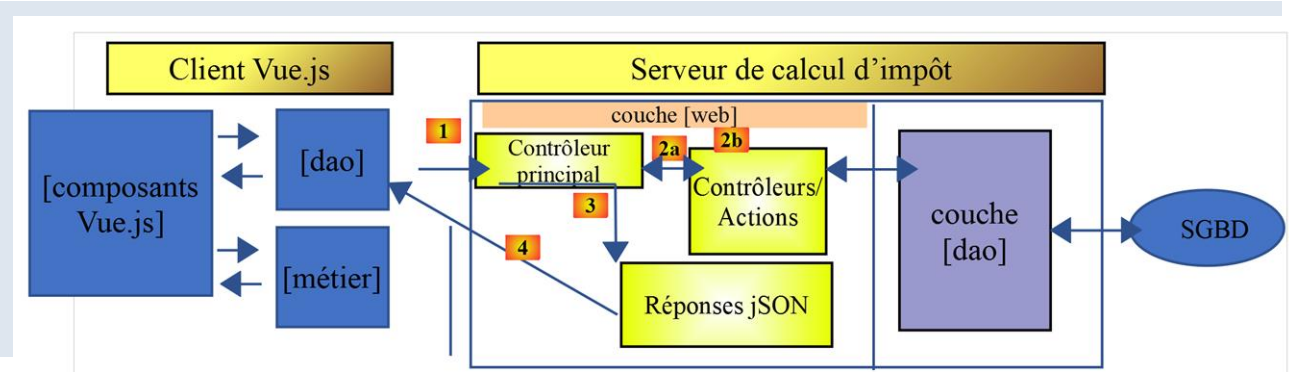
3.17.6 Conclusion

Les attributs **[Vue.$session]** du projet **[vuejs-13]** et **[Vue.$store]** du projet **[vuejs-15]** sont très proches l'un de l'autre. Ils visent le même objectif : partager de l'information entre vues. L'avantage de l'objet **[store]** est d'être réactif alors que l'objet **[session]** ne l'est pas. Mais le projet **[vuejs-14]** a montré qu'il était aisé de rendre réactif l'objet **[session]** en le dupliquant dans les propriétés réactives des vues.

3.18 Client Vue.js du serveur de calcul de l'impôt

3.18.1 Architecture

Nous allons implémenter une application client / serveur avec l'architecture suivante :

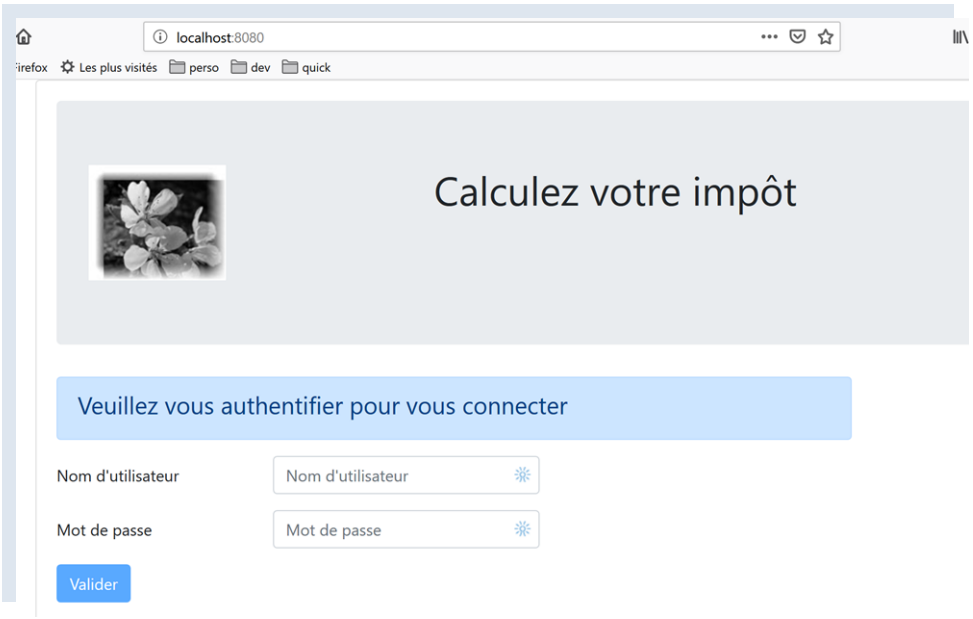


Le serveur de calcul de l'impôt sera la version 13 développée initialement dans le document |Introduction au langage PHP 7| et améliorée en une version 14 dans le document |Introduction à ECMASCRIPT 6|.

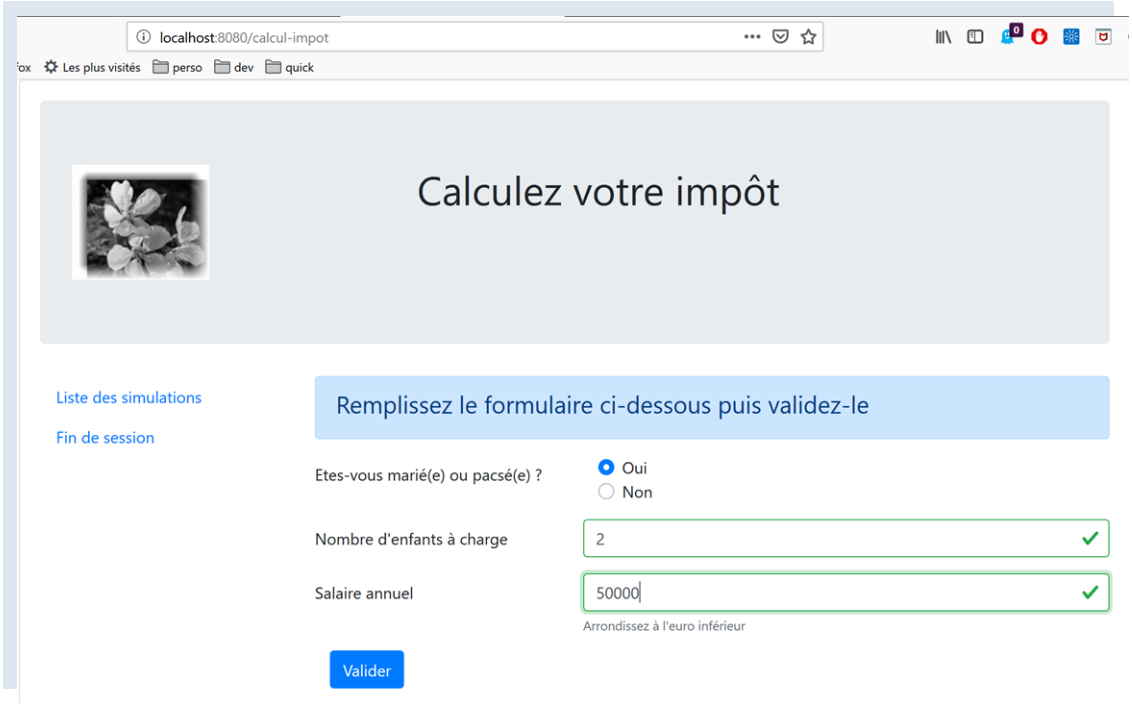
3.18.2 Les vues de l'application

Les vues de l'application **[vuejs-10]** sont celles de la version 13 du document |https://tahe.developpez.com/tutoriels-cours/php7| du serveur de calcul de l'impôt lorsqu'il est utilisé en mode HTML. Mais dans l'application présente, ces vues seront générées par le client Javascript et non par le serveur PHP.

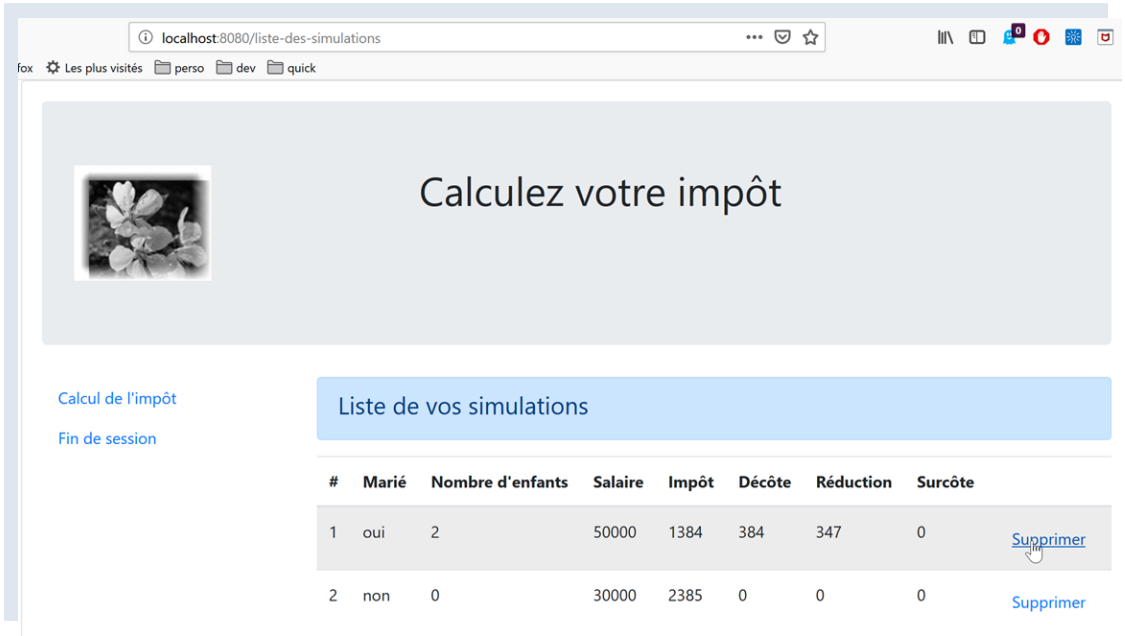
La 1ère vue est la vue d'authentification :



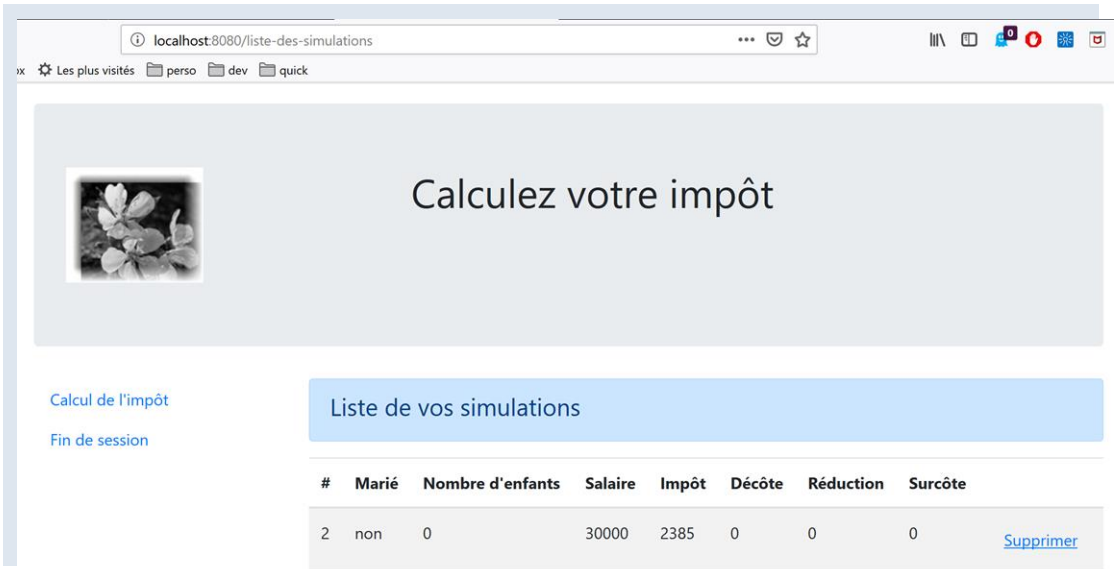
La seconde vue est celle du calcul de l'impôt :



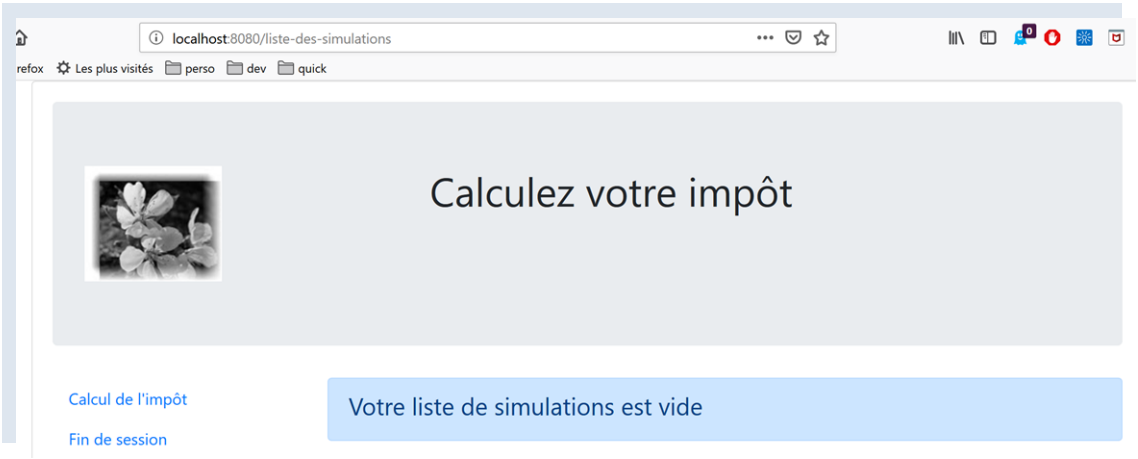
La 3ième vue est celle qui affiche la liste des simulations faites par l'utilisateur :



L'écran ci-dessus montre qu'on peut supprimer la simulation n° 1. On obtient alors la vue suivante :

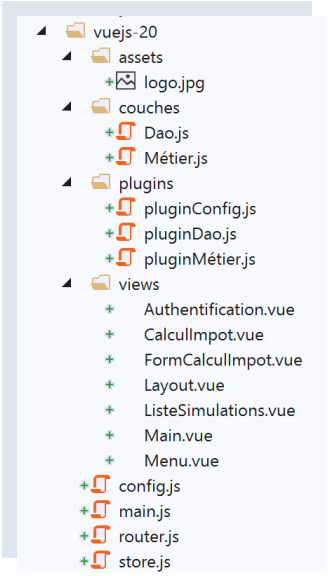


Si on supprime maintenant la dernière simulation, on obtient la nouvelle vue suivante :



3.18.3 Eléments du projet `[vuejs-20]`

L'arborescence du projet **`[vuejs-20]`** est la suivante :



Les éléments du projet sont les suivants :

- **[assets/logo.jpg]** : le logo du projet ;
- **[couches]** : les couches **[métier]** et **[dao]** de l'application ;
- **[plugins]** : les plugins de l'application ;
- **[views]** : les vues de l'application ;
- **[config.js]** : configure l'application ;
- **[router.js]** : définit le routage de l'application ;
- **[store.js]** : le store de **[Vuex]** ;
- **[main.js]** : le script principal de l'application ;

3.18.3.1 Les couches [métier] et [dao]

3.18.3.1.1 La couche [dao]

La couche **[dao]** est implémentée par la classe **[Dao]** du paragraphe |vuejs-10|

3.18.3.1.2 La couche [métier]

La couche **[métier]** est implémentée par la classe **[Métier]** (cf. paragraphe). On y a ajouté la méthode **[setTaxAdminData]** suivante :

```
1.  // constructeur
2.    constructor(taxAdmindata) {
3.      // this.taxAdminData : données de l'administration fiscale
4.      this.taxAdminData = taxAdmindata;
5.    }
6.
7.    // setter
8.    setTaxAdminData(taxAdmindata) {
9.      // this.taxAdminData : données de l'administration fiscale
10.     this.taxAdminData = taxAdmindata;
11. }
```

La méthode **[setTaxAdminData]** fait la même chose que le constructeur. Sa présence permet la séquence suivante :

1 instancier la classe **[Métier]** avec une instruction **[métier=new Métier()]** lorsqu'on veut instancier la classe mais qu'on n'a pas encore la donnée **[taxAdminData]** ;
2 puis renseigner ultérieurement sa propriété **[taxAdminData]** par une opération **[métier.setTaxAdminData(taxAdmindata)]** ;

3.18.3.2 Le fichier de configuration [config]

Le fichier **[config.js]** est le suivant :

```
1.  // utilisation de la bibliothèque [axios]
2.  const axios = require("axios");
3.  // timeout des requêtes HTTP
4.  axios.defaults.timeout = 2000;
```

```
5.  // la base des URL du serveur de calcul de l'impôt
6.  // le schéma [https] pose des problèmes à Firefox parce que le serveur de calcul
7.  // de l'impôt envoie un certificat autosigné. ok avec Chrome et Edge. Safari pas testé.
8.  // avec Firefox c'est possible en demandant l'URL ci-dessous directement et en disant à Firefox
9.  // que vous acceptez le risque d'un certificat non signé. Ensuite le client [vuejs] fonctionnera.
10.
11. // axios.defaults.baseURL = 'https://localhost/php7/scripts-web/impots/version-14';
12. axios.defaults.baseURL = "https://localhost/php7/scripts-web/impots/version-14";
13.
14. // on va utiliser des cookies
15. axios.defaults.withCredentials = true;
16.
17. // export de la configuration
18. export default {
19.   axios: axios,
20. };
```

Cette configuration est celle de la bibliothèque **[axios]** que la couche **[dao]** utilise pour faire ses requêtes HTTP. On notera ligne 8, que le serveur opère sur port sécurisé **[https]**.

3.18.3.3 Les plugins

Les plugins **[pluginDao, pluginMétier, pluginConfig]** ont pour but de créer trois nouvelles propriétés à la fonction / classe **[Vue]** :

- **[$dao]** : aura pour valeur une instance de la classe **[Dao]** ;
- **[$métier]** : aura pour valeur une instance de la classe **[Métier]** ;
- **[$config]** : aura pour valeur l'objet exporté par le fichier de configuration **[config]** ;

[pluginDao]

```
1.  export default {
2.    install(Vue, dao) {
3.      // ajoute une propriété [$dao] à la classe Vue
4.      Object.defineProperty(Vue.prototype, '$dao', {
5.        // lorsque Vue.$dao est référencé, on rend le 2ième paramètre [dao]
6.        get: () => dao,
7.      })
8.    }
9.  }
```

[pluginMétier]

```
1.  export default {
2.    install(Vue, métier) {
3.      // ajoute une propriété [$métier] à la classe Vue
4.      Object.defineProperty(Vue.prototype, '$métier', {
5.        // lorsque Vue.$métier est référencé, on rend le 2ième paramètre [métier]
6.        get: () => métier,
7.      })
8.    }
9.  }
```

[pluginConfig]

```
1.  export default {
2.    install(Vue, config) {
3.      // ajoute une propriété [$config] à la classe vue
4.      Object.defineProperty(Vue.prototype, '$config', {
5.        // lorsque Vue.$config est référencé, on rend le 2ième paramètre [config]
6.        get: () => config,
7.      })
8.    }
9.  }
```

3.18.3.4 Le store [Vuex]

Le store de **[Vuex]** est implémenté par le fichier **[store]** suivant :

```
1.  // plugin Vuex
2.  import Vue from 'vue'
3.  import Vuex from 'vuex'
4.  Vue.use(Vuex);
5.
6.  // store Vuex
```

```
7.  const store = new Vuex.Store({
8.    state: {
9.      // le tableau des simulations
10.     simulations: [],
11.     // le n° de la dernière simulation
12.     idSimulation: 0
13.   },
14.   mutations: {
15.     // suppression ligne n° index
16.     deleteSimulation(state, index) {
17.       // eslint-disable-next-line no-console
18.       console.log("mutation deleteSimulation");
19.       // on supprime la ligne n° [index]
20.       state.simulations.splice(index, 1);
21.       // eslint-disable-next-line no-console
22.       console.log("store simulations", state.simulations);
23.     },
24.     // ajout d'une simulation
25.     addSimulation(state, simulation) {
26.       // eslint-disable-next-line no-console
27.       console.log("mutation addSimulation");
28.       // n° de la simulation
29.       state.idSimulation++;
30.       simulation.id = state.idSimulation;
31.       // on ajoute la simulation au tableau des simulations
32.       state.simulations.push(simulation);
33.     },
34.     // nettoyage state
35.     clear(state) {
36.       state.simulations = [];
37.       state.idSimulation = 1;
38.     }
39.   }
40. });
41. // export de l'objet [store]
42. export default store;
1.
```

Commentaires

- lignes 2-4 : le plugin **[Vuex]** est intégré au framework **[Vue]** ;
- lignes 8-13 : nous mettons dans le store de **[Vuex]** les éléments suivants :
  - **[simulations]** : la liste des simulations faites par l'utilisateur ;
  - **[idSimulation]** : le n° de la dernière simulation faite par l'utilisateur ;
    - On rappelle que le store va être partagé entre les vues et que son contenu est réactif : lorsqu'il est modifié, les vues qui l'utilisent sont automatiquement mises à jour. Dans notre application, seul l'élément **[simulations]** a besoin d'être réactif, pas l'élément **[idSimulation]**. On a laissé cet élément dans le store par commodité ;
- lignes 14-40 : les mutations autorisées sur l'objet **[state]** des lignes 8-13. On rappelle que celles-ci reçoivent toujours l'objet **[state]** des lignes 8-13 en 1er paramètre ;
  - ligne 16 : la mutation **[deleteSimulation]** permet de supprimer une simulation dont on donne le n° **[index]** ;
  - ligne 25 : la mutation **[addSimulation]** permet d'ajouter une nouvelle simulation au tableau des simulations ;
  - ligne 35 : la mutation **[clear]** permet de réinitialiser l'objet **[state]** des lignes 8-13 ;

3.18.3.5 Le fichier de routage [router]

Le fichier de routage est le suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.  import VueRouter from 'vue-router'
4.  // les vues
5.  import Authentification from './views/Authentification'
6.  import CalculImpot from './views/CalculImpot'
7.  import ListeSimulations from './views/ListeSimulations'
8.
9.  // plugin de routage
10. Vue.use(VueRouter)
11.
12. // les routes de l'application
13. const routes = [
14.   // authentification
15.   {
16.     path: '/', name: 'authentification', component: Authentification
17.   },
```

```
18.   // calcul de l'impôt
19.   {
20.     path: '/calcul-impot', name: 'calculImpot', component: CalculImpot
21.   },
22.   // liste des simulations
23.   {
24.     path: '/liste-des-simulations', name: 'listeSimulations', component: ListeSimulations
25.   },
26.   // fin de session
27.   {
28.     path: '/fin-session', name: 'finSession', component: Authentification
29.   }
30. ]
31.
32. // le routeur
33. const router = new VueRouter({
34.   // les routes
35.   routes,
36.   // le mode d'affichage des routes dans le navigateur
37.   mode: 'history',
38. })
39.
40. // export du router
41. export default router
```

Commentaires

- ligne 16 : au démarrage de l'application, c'est la vue **[Authentification]** qui est affichée car son URL est la racine **[/]** ;
- ligne 20 : la vue **[CalculImpot]** est affichée lorsque l'URL **[/calcul-impot]** est demandée ;
- ligne 24 : la vue **[ListeSimulations]** est affichée lorsque l'URL **[/liste-des-simualtions]** est demandée ;
- ligne 28 : la vue **[Authentification]** est affichée lorsque l'URL **[/fin-session]** est demandée ;
- lignes 33-38 : un objet **[router]** est créé avec ces routes (ligne 35) et le mode **[history]** (ligne 37) de gestion des URL ;
- ligne 41 : ce routeur est exporté ;

3.18.3.6 Le script principal [main.js]

Le script **[main.js]** est le suivant :

```
1.  // imports
2.  import Vue from 'vue'
3.
4.  // vue principale
5.  import Main from './views/Main.vue'
6.
7.  // plugin [bootstrap-vue]
8.  import BootstrapVue from 'bootstrap-vue'
9.  Vue.use(BootstrapVue);
10.
11. // CSS bootstrap
12. import 'bootstrap/dist/css/bootstrap.css'
13. import 'bootstrap-vue/dist/bootstrap-vue.css'
14.
15. // routeur
16. import router from './router'
17.
18. // plugin [config]
19. import config from './config';
20. import pluginConfig from './plugins/pluginConfig'
21. Vue.use(pluginConfig, config)
22.
23. // instanciation couche [dao]
24. import Dao from './couches/Dao';
25. const dao = new Dao(config.axios);
26.
27. // plugin [dao]
28. import pluginDao from './plugins/pluginDao'
29. Vue.use(pluginDao, dao)
30.
31. // instanciation couche [métier]
32. import Métier from './couches/Métier';
33. const métier = new Métier();
34.
35. // plugin [métier]
36. import pluginMétier from './plugins/pluginMétier'
37. Vue.use(pluginMétier, métier)
```

```
38.
39. // store Vuex
40. import store from './store'
41.
42. // démarrage de l'UI
43. new Vue({
44.   el: '#app',
45.   // le routeur
46.   router: router,
47.   // le store Vuex
48.   store: store,
49.   // la vue principale
50.   render: h => h(Main),
51. })
```

On notera les points suivants :

- lignes 18-21, l'objet exporté par le script **[./config]** va être disponible dans l'attribut **[Vue.$config]** donc disponible à toutes les vues de l'application. C'était inutile ici car l'objet **[config]** n'est utilisé que par le script **[main]** (ligne 25). Néanmoins il est fréquent que la configuration soit nécessaire à plusieurs vues. On a donc voulu ici garder le principe de la rendre disponible dans un attribut de la vue ;
- lignes 24-25 : instanciation de la couche **[dao]**. La classe **[Dao]** est importée ligne 24 puis instanciée ligne 25. Son constructeur admet pour unique paramètre l'objet **[axios]**, propriété de configuration ;
- lignes 27-29 : la couche **[dao]** est rendue disponible dans l'attribut **[$dao]** de toutes les vues ;
- lignes 31-37 : on répète la même séquence pour la couche **[métier]**. Le constructeur de la classe **[Métier]** a pour paramètre **[taxAdminData]** qui représente les données de l'administration fiscale. Nous n'avons pas encore cette donnée. L'objet **[métier]** de la ligne 33 devra donc être complété ultérieurement ;
- ligne 40 : on importe le store **[Vuex]** ;
- lignes 43-51 : on instancie la vue principale **[Main]** (lignes 5 et 50), en lui passant deux paramètres :
  - ligne 46 : le routeur **[router]** défini ligne 16 ;
  - ligne 48 : le store **[Vuex] [store]** défini ligne 40 ;
  - dans les deux cas, le nom de la propriété est à gauche et sa valeur à droite. Les noms des propriétés **[router, store]** sont fixés par les frameworks **[vue-router]** et **[vuex]**. Les valeurs associées peuvent elles être quelconques ;

3.18.4 Les vues de l'application

3.18.4.1 La vue principale [Main]

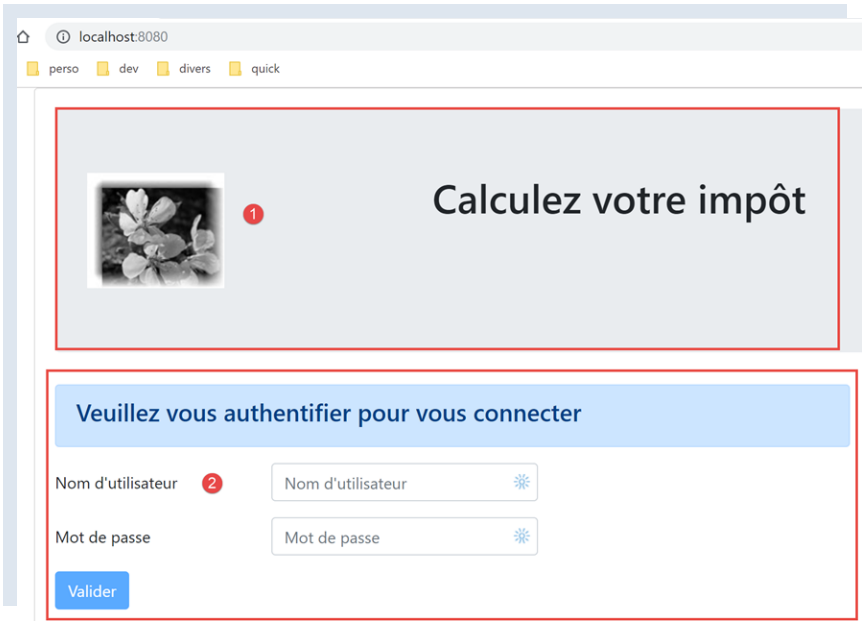
Le code de la vue principale **[Main]** est le suivant :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div class="container">
4.      <b-card>
5.        <!-- jumbotron -->
6.        <b-jumbotron>
7.          <b-row>
8.            <b-col cols="4">
9.              <img src="../assets/logo.jpg" alt="Cerisier en fleurs" />
10.           </b-col>
11.           <b-col cols="8">
12.             <h1>Calculez votre impôt</h1>
13.           </b-col>
14.         </b-row>
15.       </b-jumbotron>
16.       <!-- erreur requête HTTP -->
17.       <b-alert
18.         show
19.         variant="danger"
20.         v-if="showError"
21.       >L'erreur suivante s'est produite : {{error.message}}</b-alert>
22.       <!-- vue courante -->
23.       <router-view v-if="showView" @loading="mShowLoading" @error="mShowError" />
24.       <!-- loading -->
25.       <b-alert show v-if="showLoading" variant="light">
26.         <strong>Requête au serveur de calcul d'impôt en cours...</strong>
27.         <div class="spinner-border ml-auto" role="status" aria-hidden="true"></div>
28.       </b-alert>
29.     </b-card>
30.   </div>
31. </template>
32.
```

```
33. <script>
34. export default {
35.   // nom
36.   name: "app",
37.   // état interne
38.   data() {
39.     return {
40.       // contrôle l'alerte d'attente
41.       showLoading: false,
42.       // contrôle l'alerte d'erreur
43.       showError: false,
44.       // contrôle l'affichage de la vue de routage courante
45.       showView: true,
46.       // me message d'erreur
47.       error: ""
48.     };
49.   },
50.   // gestionnaires d'évts
51.   methods: {
52.     // erreur requête asynchrone
53.     mShowError(error) {
54.       // eslint-disable-next-line
55.       console.log("Main evt error");
56.       // on affiche le msg d'erreur
57.       this.error = error;
58.       this.showError = true;
59.       // on cache la vue routée
60.       this.showView = false;
61.       // on cache le message d'attente
62.       this.showLoading = false;
63.     },
64.     // affichage ou pas d'une icône d'attente
65.     mShowLoading(value) {
66.       // eslint-disable-next-line
67.       console.log("Main evt showLoading");
68.       // on affiche ou pas l'alerte d'attente
69.       this.showLoading = value;
70.     }
71.   }
72. };
73. </script>
```

Commentaires

- la vue **[Main]** assure une mise en page de la vue routée et affichée ligne 23 :



- les lignes 5-15 affichent la zone 1 ;
- le ligne 23 affiche la vue routée **[2]** ;
- lignes 16-19 : une alerte affichée seulement en cas d'erreur de communication avec le serveur de calcul de l'impôt ;

- lignes 25-28 : un message d'attente affiché à chaque requête HTTP faite au serveur ;
- toutes les vues vont être affichées avec cette mise en page puisque chaque vue routée est affichée par les lignes 20-24. La vue **[Main]** sert à factoriser ce qui peut être partagé par les différentes vues ;
- ligne 23 : chaque vue routée peut émettre trois événements :
  - **[loading]** : une requête HTTP a été lancée. Il faut montrer le message d'attente de la réponse ;
  - **[error]** : la requête HTTP s'est terminée sur une erreur. Il faut montrer le message d'erreur et cacher la vue routée ;
- lignes 38-49 : l'état de la vue :
  - ligne 41 : **[showLoading]** contrôle l'affichage du message d'attente de la fin d'une requête HTTP (ligne 25) ;
  - ligne 43 : **[showError]** contrôle l'affichage du message d'erreur d'une requête HTTP (lignes 17-21) ;
  - ligne 45 : **[showView]** contrôle l'affichage de la vue routée (ligne 23) ;
- lignes 53-63 : la méthode **[mShowError]** gère l'événement **[error]** émis par la vue routée (ligne 23) ;
- lignes 65-70 : la méthode **[mShowLoading]** gère l'événement **[loading]** émis par la vue routée (ligne 23) ;
- ligne 23 : on prêtera attention aux événements **[error]** et **[loading]**. Ils ne sont interceptés que si la vue routée est affichée **[showView=true]**. C'est pourquoi la vue routée est au départ affichée (ligne 45). Elle n'est cachée qu'en cas d'erreur (ligne 60). Pour éviter ce problème on aurait pu utiliser la directive **[v-show]** au lieu de **[v-if]**. la différence entre ces deux directives est la suivante :
  - **[v-if='false']** cache le bloc contrôlé en l'éliminant du code HTML global. Les événements de la vue routée ne peuvent plus alors être interceptés ;
  - **[v-show='false']** cache le bloc contrôlé en jouant sur son CSS, mais le code du bloc reste présent dans le HTML global et peut ainsi intercepter les événements de la vue routée ;

3.18.4.2 La vue de mise en page [Layout]

Le code de la vue **[Layout]** est le suivant :

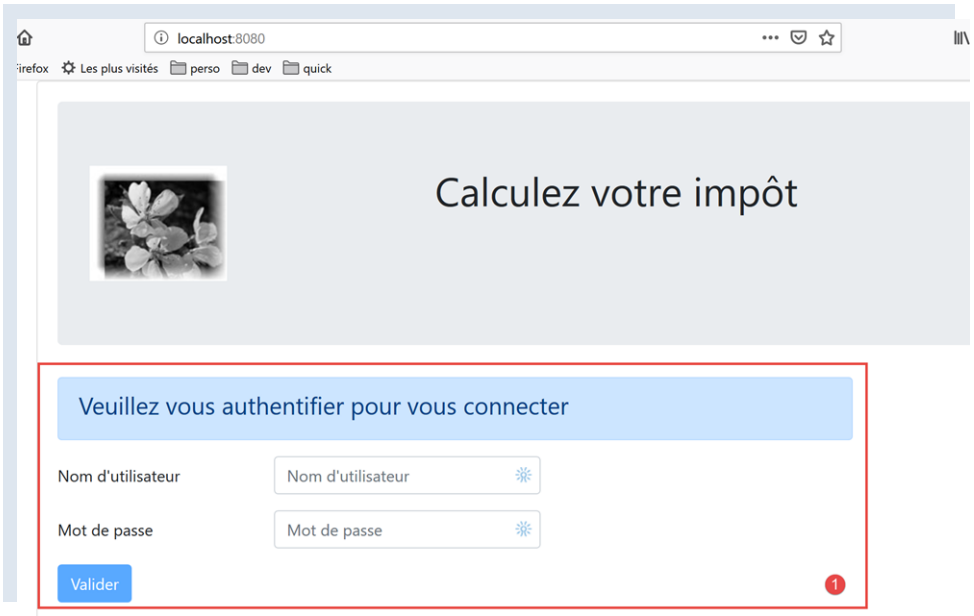
```
1.  <!-- définition HTML de la mise en page de la vue routée -->
2.  <template>
3.    <!-- ligne -->
4.    <div>
5.      <b-row>
6.        <!-- zone de trois colonnes à gauche -->
7.        <b-col cols="3" v-if="left">
8.          <slot name="left" />
9.        </b-col>
10.       <!-- zone de neuf colonnes à droite -->
11.       <b-col cols="9" v-if="right">
12.         <slot name="right" />
13.       </b-col>
14.     </b-row>
15.   </div>
16. </template>
17.
18. <script>
19.   export default {
20.     // paramètres de la vue
21.     props: {
22.       // contrôle la colonne de gauche
23.       left: {
24.         type: Boolean
25.       },
26.       // contrôle la colonne de droite
27.       right: {
28.         type: Boolean
29.       }
30.     }
31.   };
32. </script>
```

Commentaires

- la vue **[Layout]** permet de diviser la vue routée en deux zones :
  - une zone de 3 colonnes Bootstrap à gauche (lignes 7-9). Cette zone accueillera le menu de navigation lorsqu'il y en a un ;
  - une zone de 9 colonnes à droite (lignes 11-13). Cette zone accueillera l'information amenée par la vue routée ;

3.18.4.3 La vue [Authentification]

La vue d'authentification est la suivante :



Cette vue est obtenue à partir du **[Layout]** en supprimant la colonne de gauche pour n'afficher que la colonne de droite.

Son code est le suivant :

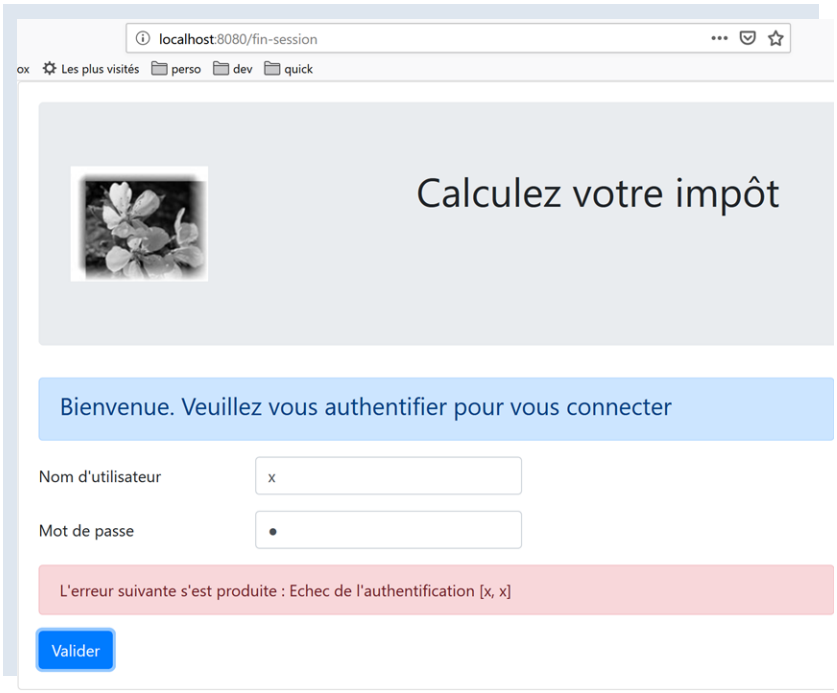
```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <Layout :left="false" :right="true">
4.      <template slot="right">
5.        <!-- formulaire HTML - on poste ses valeurs avec l'action [authentifier-utilisateur] -->
6.        <b-form @submit.prevent="login">
7.          <!-- titre -->
8.          <b-alert show variant="primary">
9.            <h4>Bienvenue. Veuillez vous authentifier pour vous connecter</h4>
10.         </b-alert>
11.         <!-- 1ère ligne -->
12.         <b-form-group label="Nom d'utilisateur" label-for="user" label-cols="3">
13.           <!-- zone de saisie user -->
14.           <b-col cols="6">
15.             <b-form-input type="text" id="user" placeholder="Nom d'utilisateur" v-model="user" />
16.           </b-col>
17.         </b-form-group>
18.         <!-- 2ième ligne -->
19.         <b-form-group label="Mot de passe" label-for="password" label-cols="3">
20.           <!-- zone de saisie password -->
21.           <b-col cols="6">
22.             <b-input type="password" id="password" placeholder="Mot de passe" v-model="password" />
23.           </b-col>
24.         </b-form-group>
25.         <!-- 3ième ligne -->
26.         <b-alert
27.           show
28.           variant="danger"
29.           v-if="showError"
30.           class="mt-3"
31.         >L'erreur suivante s'est produite : {{message}}</b-alert>
32.         <!-- bouton de type [submit] sur une 3ième ligne -->
33.         <b-row>
34.           <b-col cols="2">
35.             <b-button variant="primary" type="submit" :disabled="!valid">Valider</b-button>
36.           </b-col>
37.         </b-row>
38.       </b-form>
39.     </template>
40.   </Layout>
41. </template>
42.
43. <!-- dynamique de la vue -->
44. <script>
45. import Layout from "./Layout";
```

```
export default {
  // état du composant
  data() {
    return {
      // utilisateur
      user: "",
      // son mot de passe
      password: "",
      // contrôle l'affichage d'un msg d'erreur
      showError: false,
      // le message d'erreur
      message: "",
      // session démarrée
      sessionStarted: false
    };
  },

  // composants utilisés
  components: {
    Layout
  },

  // propriétés calculées
  computed: {
    // saisies valides
    valid() {
      return this.user && this.password && this.sessionStarted;
    }
  },

  // gestionnaires d'évts
  methods: {
    // ----------- authentification
    async login() {
      try {
        // début attente
        this.$emit("loading", true);
        // authentification bloquante auprès du serveur
        const response = await this.$dao.authentifierUtilisateur(
          this.user,
          this.password
        );
        // fin du chargement
        this.$emit("loading", false);
        // analyse de la réponse
        if (response.état != 200) {
          // on affiche l'erreur
          this.message = response.réponse;
          this.showError = true;
          return;
        }
        // pas d'erreur
        this.showError = false;
        // --------- on demande maintenant les données de l'administration fiscale
        // début attente
        this.$emit("loading", true);
        // demande bloquante auprès du serveur
        const response2 = await this.$dao.getAdminData();
        // fin du chargement
        this.$emit("loading", false);
        // analyse de la réponse
        if (response2.état != 1000) {
          // on affiche l'erreur
          this.message = response2.réponse;
          this.showError = true;
          return;
        }
        // pas d'erreur
        this.showError = false;
        // on mémorise la donnée reçue dans la couche [métier]
        this.$métier.setTaxAdminData(response2.réponse);
        // on passe à la vue du calcul de l'impôt
        this.$router.push({ name: "calculImpot" });
      } catch (error) {
        // on remonte l'erreur au composant principal
        this.$emit("error", error);
      }
```

```
123.     }
124.   },
125.   // cycle de vie : le composant vient d'être créé
126.   created() {
127.     // eslint-disable-next-line
128.     console.log("authentification", "created");
129.     // on démarre une session jSON avec le serveur
130.     // début attente
131.     this.$emit("loading", true);
132.     // on initialise la session avec le serveur - requête asynchrone
133.     // on utilise la promesse rendue par les méthodes de la couche [dao]
134.     this.$dao
135.       // on initialise une session jSON
136.       .initSession()
137.       // on a obtenu la réponse
138.       .then(response => {
139.         // fin attente
140.         this.$emit("loading", false);
141.         // analyse de la réponse
142.         if (response.état != 700) {
143.           // on affiche l'erreur
144.           this.message = response.réponse;
145.           this.showError = true;
146.           return;
147.         }
148.         // la session a démarré
149.         this.sessionStarted = true;
150.       })
151.       // en cas d'erreur
152.       .catch(error => {
153.         // on remonte l'erreur à la vue [Main]
154.         this.$emit("error", error);
155.       });
156.   }
157. };
158. </script>
```

Commentaires

- ligne 3 : la vue **[Authentification]** utilise uniquement la colonne de droite du **[Layout]** (lignes 3 et 4) ;
- lignes 6-38 : le formulaire Bootstrap qui génère la zone 1 de la copie d'écran ci-dessus ;
- ligne 6 : l'événement **[@submit]** se produit lorsque l'utilisateur va cliquer sur le bouton de type **[submit]** de la ligne 35. Le modificateur **[prevent]** demande à ce que la page ne soit pas rechargée lors du **[submit]**. On aurait pu écrire également :
  - une balise <b-form> sans gestion de l'événement **[submit]** ;
  - une balise <b-button> avec l'événement **[@click='login']** et sans l'attribut **[type='submit']** ;
    - Ca marche également. L'avantage de la solution retenue est que le submit se fait non seulement avec un clic sur le bouton **[Valider]** mais également sur une validation (touche **[Entrée]**) dans les zones de saisie. C'est donc par commodité pour l'utilisateur que la solution **[<b-form @submit.prevent="login">]** a été retenue ici ;
- lignes 33-37 : une alerte qui apparaît lorsque le serveur a rejeté les identifiants saisis par l'utilisateur :



- ligne 35 : le bouton **[Valider]** n'est pas toujours actif. Son état dépend de l'attribut calculé **[valid]** des lignes 71-73. L'attribut **[valid]** est vrai si :
  - il y a quelque chose dans les champs **[user, password]** du formulaire ;
  - la session jSON a démarré. Au départ, cette session n'a pas démarré (ligne 59) et donc le bouton **[Valider]** est inactif.
- lignes 49-60 : l'état de la vue ;
  - **[user]** représente la saisie de l'utilisateur dans le champ **[user]** (lignes 12-17) du formulaire. La directive **[v-model]** de la ligne 15 établit une liaison bidirectionnelle entre la saisie de l'utilisateur et l'attribut **[user]** de la vue ;
  - **[password]** représente la saisie de l'utilisateur dans le champ **[password]** (lignes 19-24) du formulaire. La directive **[v-model]** de la ligne 22 établit une liaison bidirectionnelle entre la saisie de l'utilisateur et l'attribut **[password]** de la vue ;
  - **[showError]** contrôle (ligne 29) l'affichage de l'alerte des lignes 26-31 ;
  - **[message]** est le message d'erreur (ligne 31) à afficher dans l'alerte des lignes 26-31 ;
  - **[sessionStarted]** indique si la session jSON avec le serveur a démarré ou non. Au départ cet attribut a la valeur **[false]** (ligne 59). La session jSON avec le serveur est initialisée dans l'événement **[created]** du cycle de vie de la vue, lignes 126-156. Si le serveur répond positivement, alors l'attribut **[sessionStarted]** est passé à **[true]** (ligne 149) ;
- lignes 126-156 : la fonction **[created]** est exécutée lorsque la vue **[Authentification]** a été créée (pas forcément encore affichée). En tâche de fond, on initialise alors une session jSON avec le serveur. On sait que c'est la 1ère action à faire avec le serveur de calcul de l'impôt. Pour ce faire, on utilise la couche **[dao]** de l'application (ligne 134). Toutes les méthodes de cette couche sont asynchrones. On utilise ici la promesse (Promise) rendue par la méthode **[$dao.initSession]** qui initialise la session jSON avec le serveur.
- lignes 138-150 : le code exécuté lorsque le serveur a rendu sa réponse sans erreur ;
- ligne 142 : on vérifie la propriété **[état]** de la réponse. Elle doit avoir la valeur **[700]** pour une opération réussie. Sinon, il s'est produit une erreur dont la cause est indiquée dans la propriété **[response.réponse]** (ligne 144). On affiche alors le message d'erreur de la vue (ligne 145) ;
- ligne 149 : on note que la session jSON a démarré ;
- lignes 152-155 : le code exécuté en cas d'erreur. Celle-ci est remontée à la vue parente **[Main]** qui
  - affichera l'erreur ;
  - cachera le message d'attente ;
  - cachera la vue routée, la vue **[Autentification]** ;
- lignes 79-124 : la méthode **[login]** traite le clic sur le bouton **[Valider]** ;
- ligne 79 : la méthode a été préfixée avec le mot clé **[async]** pour permettre l'utilisation du mot clé **[await]**, lignes 84 et 103 ;
- lignes 84-87 : appel bloquant à la méthode **[$dao.authentifierUtilisateur(user, password)]**. On aurait pu utiliser une promesse **[Promise]** comme il a été fait dans la fonction **[created]**. Nous avons voulu varier les styles. Il n'y a pas de risque à bloquer l'utilisateur car nous avons mis un **[timeout]** de 2 secondes à toutes les requêtes HTTP. Il n'attendra pas longtemps. De plus, il ne peut rien faire tant que le serveur n'a pas rendu sa réponse car alors le bouton **[Valider]** reste inactif ;
- ligne 91 : le serveur de calcul de l'impôt envoie des réponses jSON ayant toutes la structure **[{'action':action, 'état':val, 'réponse':réponse}]**. L'authentification a réussi si **[état==200]**. Si ce n'est pas le cas, un message d'erreur est affiché, lignes 93-94 ;
- ligne 98 : on cache un éventuel message d'erreur d'une opération précédente ;

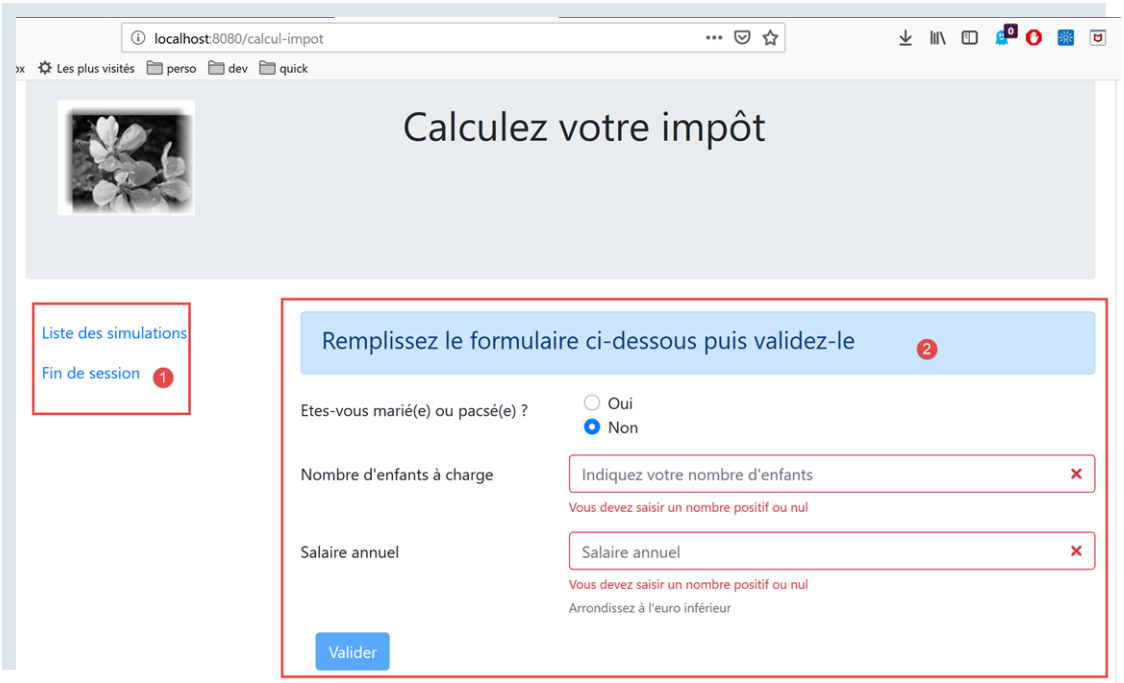
- lignes 99-116 : on demande mainteant au serveur les données de l'administration fiscale qui permettent le calcul de l'impôt. Dans **[this.$métier]** nous avons une instance de la classe **[Métier]** qui pour l'instant ne peut rien faire car elle n'a pas ces données ;
- ligne 103 : les données de l'administration fiscale sont demandées au serveur par une opération bloquante ;
- lignes 107-112 : la réponse du serveur est analysée. Elle doit avoir une valeur d'état égale à 1000 sinon c'est qu'il s'est produit une erreur. Dans ce dernier cas, on affiche le message d'erreur (lignes 109-110) ;
- lignes 113-118 : en cas de réussite de l'opération, on :
  - cache le message d'erreur, ligne 114 ;
  - on transmet les données de l'administration fiscale à la couche **[métier]** (ligne 116) ;
  - on fait afficher la vue **[CalculImpot]**, ligne 118. On se rappelle que **[this.$router]** désigne le routeur de l'application. La méthode **[push]** permet de fixer la prochaine vue routée. Ici on la désigne par son attribut **[name]**. On aurait pu également la désigner par son attribut **[path]**. Ces informations sont dans le fichier de routage :

```
1.  // calcul de l'impôt
2.    {
3.      path: '/calcul-impot', name: 'calculImpot', component: CalculImpot
4.  },
```

- lignes 119-122 : le **[catch]** se déclenche lorsqu'une des deux requêtes HTTP a échoué (serveur pas présent, timeout dépassé, ...). On signale alors l'erreur à la vue parente **[Main]** qui l'affichera, cachera le message d'attente et la vue **[Authentification]** ;

3.18.4.4 La vue [CalculImpot]

La vue **[CalculImpot]** est la suivante :



- **[1]** : un menu de navigation occupe la colonne de gauche de la vue routée ;
- **[2]** : le formulaire de calcul de l'impôt occupe la colonne de droite de la vue routée ;

Le code de la vue **[CalculImpot]** est le suivant :

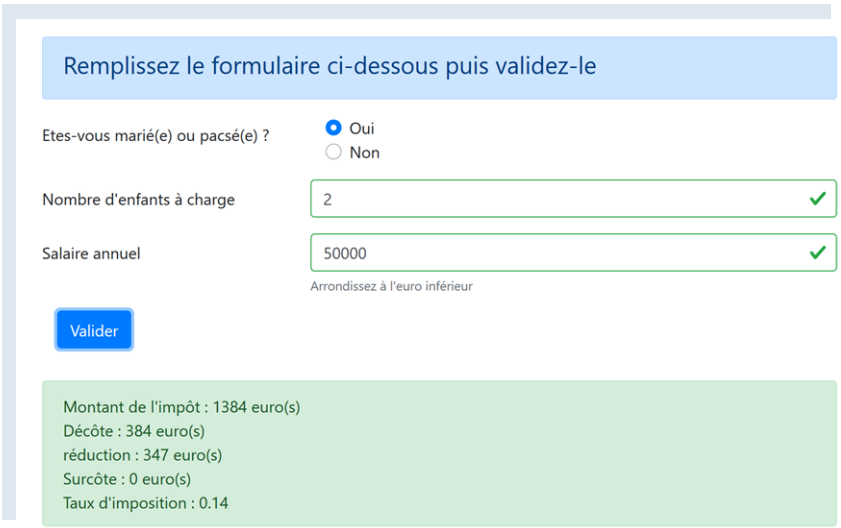
```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <Layout :left="true" :right="true">
5.        <!-- formulaire de calcul de l'impôt à droite -->
6.        <FormCalculImpot slot="right" @resultatObtenu="handleResultatObtenu" />
7.        <!-- menu de navigation à gauche -->
8.        <Menu slot="left" :options="options" />
9.      </Layout>
10.     <!-- zone d'affichage des résultat du calcul de l'impôt sous le formulaire -->
11.     <b-row v-if="résultatObtenu" class="mt-3">
12.       <!-- zone de trois colonnes vide -->
13.       <b-col cols="3" />
```

```
14.       <!-- zone de neuf colonnes -->
15.       <b-col cols="9">
16.         <b-alert show variant="success">
17.           <span v-html="résultat"></span>
18.         </b-alert>
19.       </b-col>
20.     </b-row>
21.   </div>
22. </template>
23.
24. <script>
25. // imports
26. import FormCalculImpot from "./FormCalculImpot";
27. import Menu from "./Menu";
28. import Layout from "./Layout";
29.
30. export default {
31.   // état interne
32.   data() {
33.     return {
34.       // options du menu
35.       options: [
36.         {
37.           text: "Liste des simulations",
38.           path: "/liste-des-simulations"
39.         },
40.         {
41.           text: "Fin de session",
42.           path: "/fin-session"
43.         }
44.       ],
45.       // résultat du calcul de l'impôt
46.       résultat: "",
47.       résultatObtenu: false
48.     };
49.   },
50.   // composants utilisés
51.   components: {
52.     Layout,
53.     FormCalculImpot,
54.     Menu
55.   },
56.   // méthodes de gestion des évts
57.   methods: {
58.     // résultat du calcul de l'impôt
59.     handleResultatObtenu(résultat) {
60.       // on construit le résultat en chaîne HTML
61.       const impôt = "Montant de l'impôt : " + résultat.impôt + " euro(s)";
62.       const décôte = "Décôte : " + résultat.décôte + " euro(s)";
63.       const réduction = "Réduction : " + résultat.réduction + " euro(s)";
64.       const surcôte = "Surcôte : " + résultat.surcôte + " euro(s)";
65.       const taux = "Taux d'imposition : " + résultat.taux;
66.       this.résultat =
67.         impôt +
68.         "<br/>" +
69.         décôte +
70.         "<br/>" +
71.         réduction +
72.         "<br/>" +
73.         surcôte +
74.         "<br/>" +
75.         taux;
76.       // affichage du résultat
77.       this.résultatObtenu = true;
78.       // ---- maj du store [Vuex]
79.       // une simulation de +
80.       this.$store.commit("addSimulation", résultat);
81.     }
82.   }
83. };
84. </script>
```

Commentaires

- ligne 4 : les deux colonnes du **[Layout]** sont ici présentes ;

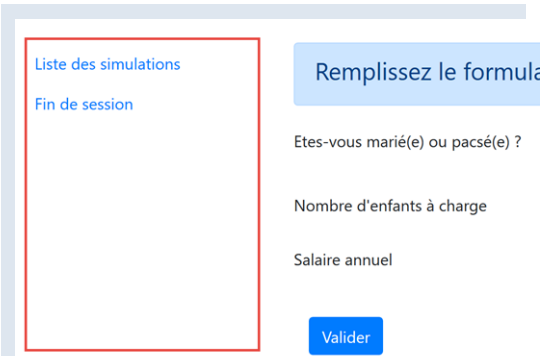
- ligne 6 : le formulaire de calcul de l'impôt occupe la colonne de droite. Il émet l'événement **[resultatObtenu]** lorsque le résultat du calcul de l'impôt a été obtenu. On notera que les noms d'événements et les noms des méthodes qui les gèrent ne peuvent contenir de caractères accentués ;
- ligne 8 : le menu de navigation occupe la colonne de gauche ;
- lignes 11-20 : le résultat du calcul de l'impôt est affiché sous le formulaire :



- ligne 11 : le résultat n'est affiché que si l'attribut **[résultatObtenu]** (ligne 47) vaut **[true]** ;
- lignes 34-48 : l'état de la vue :
  - **[options]** : la liste des options du menu de navigation. Ce tableau est passé en paramètre au composant **[Menu]**, ligne 8 ;
  - **[résultat]** : le résultat du calcul de l'impôt. Ce résultat est une chaîne HTML. C'est pourquoi on a utilisé la directive **[v-html]** à la ligne 17 pour l'afficher ;
  - **[résultatObtenu]** : le booléen qui contrôle l'affichage du résultat, ligne 11 ;
- lignes 59-81 : la méthode **[handleResultatObtenu]** affiche le résultat du calcul de l'impôt que lui a envoyé la vue fille **[FormCalculImpot]**, ligne 6. Ce résultat est un objet avec les propriétés **[impot, décôte, réduction, surcôte, taux, marié, enfants, salaire]** ;
- lignes 61-75 : on inscrit l'objet **[impot, décôte, réduction, surcôte, taux]** dans un texte HTML qui est visualisé par la ligne 17 du template ;
- ligne 77 : on affiche ce résultat ;
- ligne 80 : on appelle la mutation **[addSimulation]** du store Vuex qui va ajouter **[résultat]** aux simulations déjà présentes dans le store ;

3.18.4.5 Le menu de navigation [Menu]

Le menu de navigation s'affiche dans la colonne de gauche des vues routées :



Le code de la vue **[Menu]** est le suivant :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- menu Bootstrap vertical -->
4.    <b-nav vertical>
5.      <!-- options du menu -->
6.      <b-nav-item
```

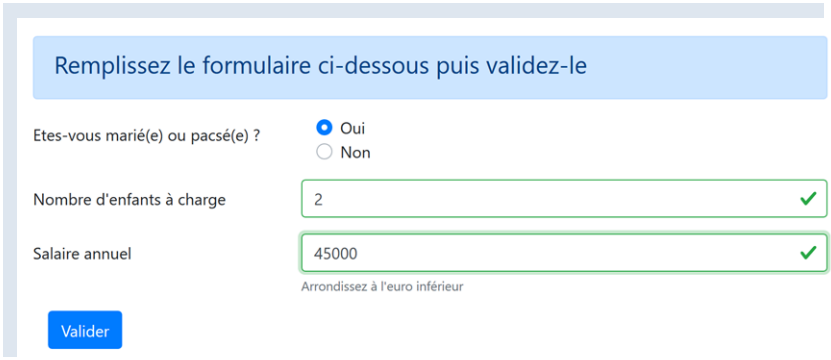
```
7.        v-for="(option,index) of options"
8.        :key="index"
9.        :to="option.path"
10.       exact
11.       exact-active-class="active"
12.     >{{option.text}}</b-nav-item>
13.   </b-nav>
14. </template>
15.
16. <script>
17. export default {
18.   // paramètres de la vue
19.   props: {
20.     options: {
21.       type: Array
22.     }
23.   }
24. };
25. </script>
```

Commentaires

- les options du menu sont fournies par le paramètre **[options]** (lignes 7, 20-22) ;
- chaque élément du tableau **[options]** a une propriété **[text]** (ligne 12) qui est le texte du lien et une propriété **[path]** (ligne 9) qui sera le chemin de la vue cible du lien ;

3.18.4.6 La vue [FormCalculImpot]

Cette vue fournit le formulaire de calcul de l'impôt :



Son code est le suivant :

```
1.    <!-- définition HTML de la vue -->
2.    <template>
3.    <!-- formulaire HTML -->
4.    <b-form @submit.prevent="calculerImpot" class="mb-3">
5.      <!-- message sur 12 colonnes sur fond bleu -->
6.      <b-alert show variant="primary">
7.        <h4>Remplissez le formulaire ci-dessous puis validez-le</h4>
8.      </b-alert>
9.      <!-- éléments du formulaire -->
10.     <!-- première ligne -->
11.     <b-form-group label="Etes-vous marié(e) ou pacsé(e) ?" label-cols="4">
12.       <!-- boutons radio sur 5 colonnes-->
13.       <b-col cols="5">
14.         <b-form-radio v-model="marié" value="oui">Oui</b-form-radio>
15.         <b-form-radio v-model="marié" value="non">Non</b-form-radio>
16.       </b-col>
17.     </b-form-group>
18.     <!-- deuxième ligne -->
19.     <b-form-group label="Nombre d'enfants à charge" label-cols="4" label-for="enfants">
20.       <b-input
21.         type="text"
22.         id="enfants"
23.         placeholder="Indiquez votre nombre d'enfants"
24.         v-model="enfants"
25.         :state="enfantsValide"
26.       />
27.       <!-- message d'erreur éventuel -->
```

```
28.       <b-form-invalid-feedback :state="enfantsValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
29.     </b-form-group>
30.     <!-- troisème ligne -->
31.     <b-form-group
32.       label="Salaire annuel"
33.       label-cols="4"
34.       label-for="salaire"
35.       description="Arrondissez à l'euro inférieur"
36.     >
37.       <b-input
38.         type="text"
39.         id="salaire"
40.         placeholder="Salaire annuel"
41.         v-model="salaire"
42.         :state="salaireValide"
43.       />
44.       <!-- message d'erreur éventuel -->
45.       <b-form-invalid-feedback :state="salaireValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
46.     </b-form-group>
47.     <!-- quatrième ligne, bouton [submit] sur 5 colonnes -->
48.     <b-col cols="5">
49.       <b-button type="submit" variant="primary" :disabled="formInvalide">Valider</b-button>
50.     </b-col>
51.   </b-form>
52. </template>
53.
54. <!-- script -->
55. <script>
56. export default {
57.   // état interne
58.   data() {
59.     return {
60.       // marié ou pas
61.       marié: "non",
62.       // nombre d'enfants
63.       enfants: "",
64.       // salaire annuel
65.       salaire: ""
66.     };
67.   },
68.   // état interne calculé
69.   computed: {
70.     // validation du formulaire
71.     formInvalide() {
72.       return (
73.         // salaire invalide
74.         !this.salaireValide ||
75.         // ou enfants invalide
76.         !this.enfantsValide ||
77.         // ou données fiscales pas obtenues
78.         !this.$métier.taxAdminData
79.       );
80.     },
81.     // validation du salaire
82.     salaireValide() {
83.       // doit être numérique >=0
84.       return Boolean(this.salaire.match(/^\s*\d+\s*$/));
85.     },
86.     // validation des enfants
87.     enfantsValide() {
88.       // doit être numérique >=0
89.       return Boolean(this.enfants.match(/^\s*\d+\s*$/));
90.     }
91.   },
92.   // gestionnaire d'évts
93.   methods: {
94.     calculerImpot() {
95.       // on calcule l'impôt à l'aide de la couche [métier]
96.       const résultat = this.$métier.calculerImpot(
97.         this.marié,
98.         this.enfants,
99.         this.salaire
100.      );
101.      // eslint-disable-next-line
102.      console.log("résultat=", résultat);
```

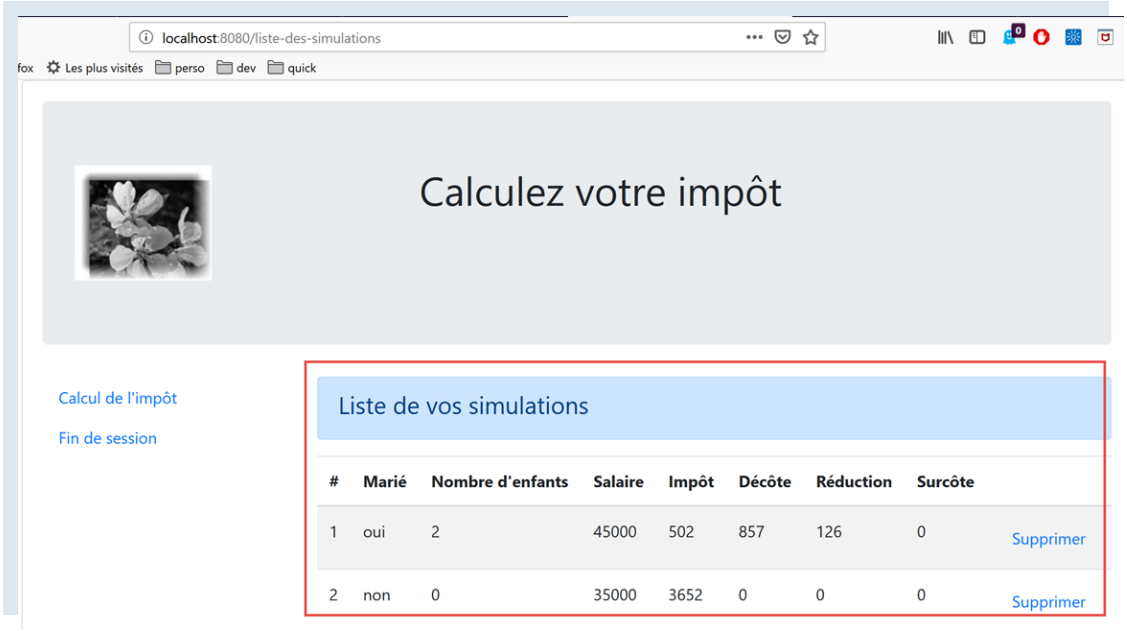
```
103.       // on complète le résultat
104.       résultat.marié = this.marié;
105.       résultat.enfants = this.enfants;
106.       résultat.salaire = this.salaire;
107.       // on émet l'évt [resultatObtenu]
108.       this.$emit("resultatObtenu", résultat);
109.     }
110.   }
111.};
112.</script>
```

Commentaires

- lignes 4-51 : le formulaire Bootstrap ;
- lignes 11-17 : un groupe de boutons radio avec leur libellé ;
- lignes 14-15 : la balise <b-form-radio> assure l'affichage d'un bouton radio :
  - ligne 14 : la directive **[v-model]** assure que lors d'un clic sur le bouton, l'attribut **[marié]** de la ligne 61 recevra la valeur **[oui]** (attribut **[value="oui"]**) ;
  - ligne 15 : la directive **[v-model]** assure que lors d'un clic sur le bouton, l'attribut **[marié]** de la ligne 61 recevra la valeur **[non]** (attribut **[value="non"]**) ;
- lignes 19-29 : la partie saisie du nombre d'enfants :
  - ligne 24 : la saisie du nombre d'enfants est liée à l'attribut **[enfants]** de la ligne 63 ;
  - ligne 25 : la validité de la saisie est vérifiée par l'attribut calculé **[enfantsValide]** des lignes 87-89 ;
  - ligne 28 : assure l'affichage d'un message d'erreur si la saisie est invalide ;
- lignes 31-45 : la partie saisie du salaire annuel :
  - ligne 35 : affiche un message d'aide juste sous la zone de saisie ;
  - ligne 41 : la saisie du salaire est liée à l'attribut **[salaire]** de la ligne 65 ;
  - ligne 42 : la validité de la saisie est vérifiée par l'attribut calculé **[salaireValide]** des lignes 82-85 ;
  - ligne 45 : assure l'affichage d'un message d'erreur si la saisie est invalide ;
- lignes 48-50 : un bouton de type **[submit]**. Lorsqu'on clique sur ce bouton ou lorsqu'on valide une saisie avec la touche **[Entrée]**, la méthode **[calculerImpot]** est exécutée (ligne 94) ;
  - ligne 49 : l'état du bouton actif / inactif est contrôlé par l'attribut calculé **[formInvalide]** des lignes 71-80 ;
- lignes 71-80 : le formulaire est valide si :
  - le nombre d'enfants est valide ;
  - le salaire est valide ;
  - l'application a obtenu du serveur les données de l'administration fiscale permettant le calcul de l'impôt. On rappelle que cette donnée est enregistrée dans la propriété **[$métier.taxAdminData]**. La vue **[FormCalculImpot]** peut être affichée avant que cette donnée ait été obtenue car elle est demandée de façon asynchrone en même temps que se produit l'affichage de la vue. On s'assure ici que l'utilisateur ne peut pas cliquer sur le bouton **[Valider]** tant que la donnée n'a pas été obtenue ;
- lignes 94-109 : la méthode de calcul de l'impôt :
  - lignes 96-100 : c'est la couche **[métier]** qui fait ce calcul. C'est un calcul synchrone. Une fois la donnée **[taxAdminData]** a été obtenue, le client **[Vue]** n'a plus à communiquer avec le serveur. Tout se fait localement. On obtient un objet **[résultat]** avec les propriétés **[impôt, décôte, surcôte, réduction, taux]** ;
  - lignes 104-106 : on ajoute les propriétés **[marié, enfants, salaire]** au résultat ;
  - ligne 108 : le résultat est passé à la vue parent **[CalculImpot]** via l'événement **[resultatObtenu]**. Cette vue est chargée d'afficher le résultat ;

3.18.4.7 La vue [ListeSimulations]

La vue **[ListeSimulations]** affiche la liste des simulations faites par l'utilisateur :



Le code de la vue est le suivant :

```
<!-- définition HTML de la vue -->
<template>
  <div>
    <!-- mise en page -->
    <Layout :left="true" :right="true">
      <!-- simulations dans colonne de droite -->
      <template slot="right">
        <template v-if="simulations.length==0">
          <!-- pas de simulations -->
          <b-alert show variant="primary">
            <h4>Votre liste de simulations est vide</h4>
          </b-alert>
        </template>
        <template v-if="simulations.length!=0">
          <!-- il y a des simulations -->
          <b-alert show variant="primary">
            <h4>Liste de vos simulations</h4>
          </b-alert>
          <!-- tableau des simulations -->
          <b-table striped hover responsive :items="simulations" :fields="fields">
            <template v-slot:cell(action)="data">
              <b-button variant="link" @click="supprimerSimulation(data.index)">Supprimer</b-button>
            </template>
          </b-table>
        </template>
      </template>
      <!-- menu de navigation dans colonne de gauche -->
      <Menu slot="left" :options="options" />
    </Layout>
  </div>
</template>

<script>
// imports
import Layout from "./Layout";
import Menu from "./Menu";
export default {
  // composants
  components: {
    Layout,
    Menu
  },
  // état interne
  data() {
    return {
      // options du menu de navigation
      options: [
```

```
48.         {
49.           text: "Calcul de l'impôt",
50.           path: "/calcul-impot"
51.         },
52.         {
53.           text: "Fin de session",
54.           path: "/fin-session"
55.         }
56.       ],
57.       // paramètres de la table HTML
58.       fields: [
59.         { label: "#", key: "id" },
60.         { label: "Marié", key: "marié" },
61.         { label: "Nombre d'enfants", key: "enfants" },
62.         { label: "Salaire", key: "salaire" },
63.         { label: "Impôt", key: "impôt" },
64.         { label: "Décôte", key: "décôte" },
65.         { label: "Réduction", key: "réduction" },
66.         { label: "Surcôte", key: "surcôte" },
67.         { label: "", key: "action" }
68.       ]
69.     };
70.   },
71.   // état interne calculé
72.   computed: {
73.     // liste des simulations prise dans le store Vuex
74.     simulations() {
75.       return this.$store.state.simulations;
76.     }
77.   },
78.   // méthodes
79.   methods: {
80.     supprimerSimulation(index) {
81.       // eslint-disable-next-line
82.       console.log("supprimerSimulation", index);
83.       // suppression de la simulation n° [index]
84.       this.$store.commit("deleteSimulation", index);
85.     }
86.   }
87. };
88. </script>
```

Commentaires

- ligne 5 : la vue occupe les deux colonnes de la mise en page **[Layout]** des vues routées ;
- lignes 7-26 : les simulations vont dans la colonne de droite ;
- ligne 28 : le menu de navigation va dans la colonne de gauche ;
- lignes 8, 14, 20, 75 : les simulations proviennent du store **[Vuex]** **[$this.store]** ;
- lignes 8-13 : alerte affichée lorsque la liste des simulations est vide ;
- lignes 14-25 : la table HTML affichée lorsque la liste des simulations n'est pas vide ;
- lignes 20-24 : la table HTML est générée par une balise <b-table> ;
  - ligne 20 : le tableau des simulations est fourni par l'attribut calculé **[simulations]** des lignes 74-76 ;
  - ligne 20 : la configuration de la table HTML est faite par l'attribut calculé **[fields]** des lignes 58-69. Ligne 67, la colonne de clé **[action]** est la dernière colonne de la table HTML ;
  - lignes 21-23 : template de la dernière colonne de la table HTML ;
  - ligne 22 : on y met un bouton de type lien. Lorsqu'on clique dessus, la méthode **[supprimerSimulation(data.index)]** est appelée, où **[data]** représente la ligne courante (ligne 21). **[data.index]** représente le n° de cette ligne dans la liste des lignes affichées ;
- ligne 28 : génération du menu de navigation. Les options de celui-ci sont fournies par l'attribut **[options]** des lignes 47-56 ;
- lignes 80-85 : la méthode qui réagit au clic sur un lien **[Supprimer]** de la page HTML ;
  - ligne 84 : on fait appel à la mutation **[deleteSimulation]** du store **[Vuex]** (cf paragraphe |vuejs-15|) ;

3.18.5 Exécution du projet

Il faut également lancer le serveur **[Laragon]** (cf document |https://tahe.developpez.com/tutoriels-cours/php7|) pour que le serveur de calcul d'impôt soit en ligne.

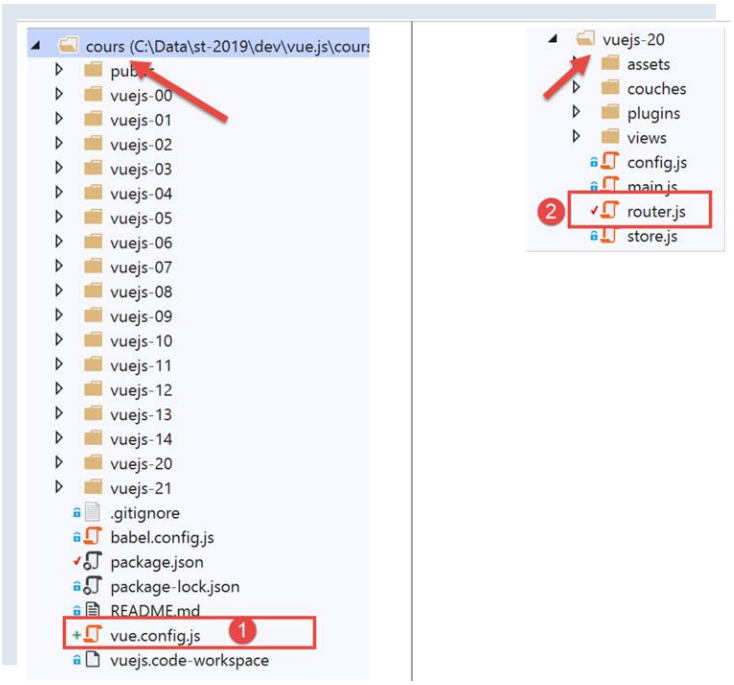
3.18.6 Déploiement de l'application sur un serveur local

Actuellement, notre client **[Vue]** est déployé sur un serveur de test à l'URL **[http://localhost:8080]**. Nous allons le déployer sur le serveur **[Laragon]** à l'URL **[http://localhost:80]**. Il y a plusieurs étapes à effectuer pour en arriver là.

étape 1

Tout d'abord, nous allons faire en sorte que le client **[Vue]** soit déployé sur le serveur de test à l'URL **[http://localhost:8080/client-vuejs-impot/]**.

Nous créons un fichier **[vue.config.js]** à la racine de notre projet **[VSCode]** actuel :



Le fichier **[vue.config.js]** **[1]** aura le contenu suivant :

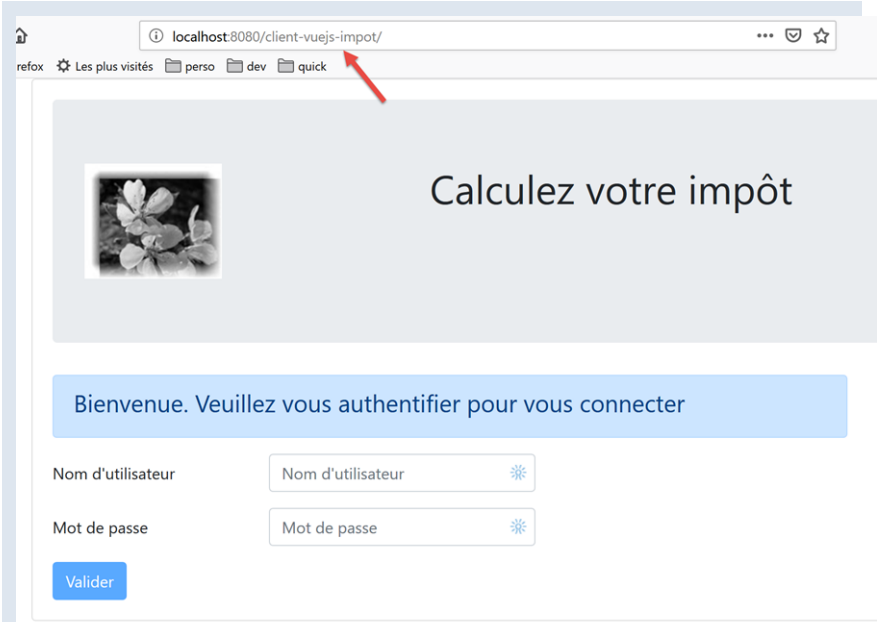
```
1.  // vue.config.js
2.  module.exports = {
3.    // l'URL de service du client [vuejs] du serveur de calcul de l'impôt
4.    publicPath: "/client-vuejs-impot/",
5.  };
```

Il nous faut également modifier le fichier de routage **[router.js]** **[2]** :

```
1.  // imports
2.  import Vue from 'vue'
3.  import VueRouter from 'vue-router'
4.  // les vues
5.  import Authentification from './views/Authentification'
6.  import CalculImpot from './views/CalculImpot'
7.  import ListeSimulations from './views/ListeSimulations'
8.
9.  // plugin de routage
10. Vue.use(VueRouter)
11.
12. // les routes de l'application
13. const routes = [
14.   // authentification
15.   {
16.     path: '/', name: 'authentification', component: Authentification
17.   },
18.   // calcul de l'impôt
19.   {
20.     path: '/calcul-impot', name: 'calculImpot', component: CalculImpot
21.   },
22.   // liste des simulations
23.   {
24.     path: '/liste-des-simulations', name: 'listeSimulations', component: ListeSimulations
25.   },
26.   // fin de session
27.   {
28.     path: '/fin-session', name: 'finSession', component: Authentification
29.   }
30. ]
31.
32. // le routeur
33. const router = new VueRouter({
34.   // les routes
35.   routes,
36.   // le mode d'affichage des routes dans le navigateur
37.   mode: 'history',
38.   // l'URL de base de l'application
39.   base: '/client-vuejs-impot/'
40. })
41.
42. // export du router
43. export default router
```

- ligne 39 : on indique au routeur que les chemins des routes définies lignes 13-30 sont relatives au chemin défini ligne 39. Par exemple, le chemin de la ligne 20 **`[/calcul-impot]`** deviendra **`[/client-vuejs-impot/calcul-impot]`** ;

On peut alors tester de nouveau le projet **`[vuejs-20]`** pour vérifier le changement des chemins de l'application :

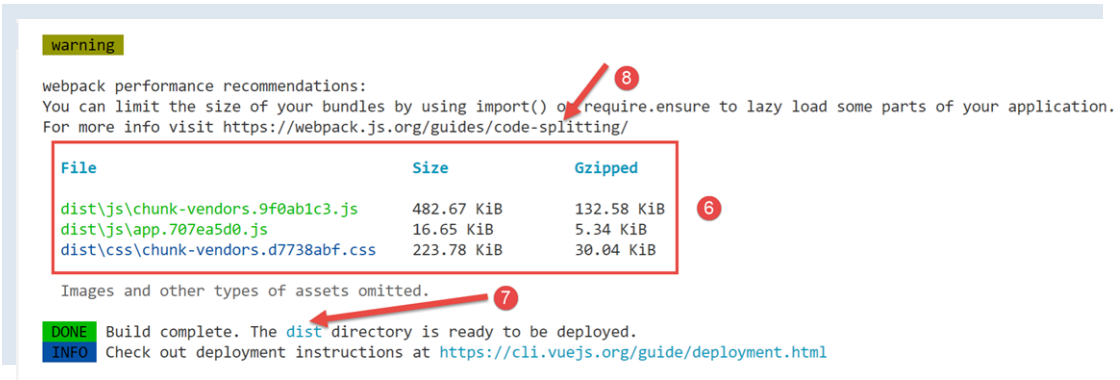


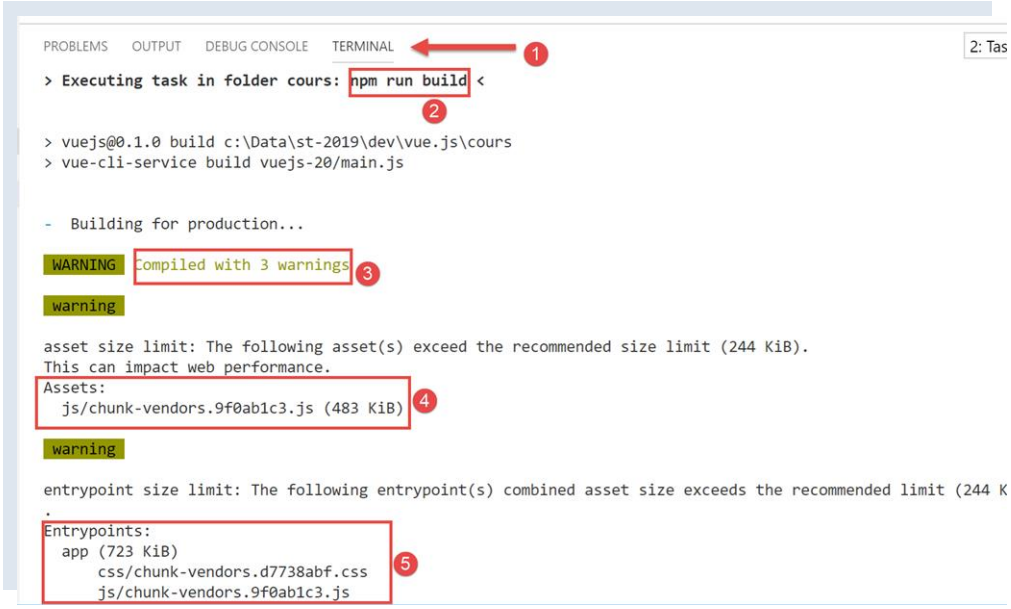
étape 2

Nous construisons maintenant la version de production du projet **[vuejs-20]** :

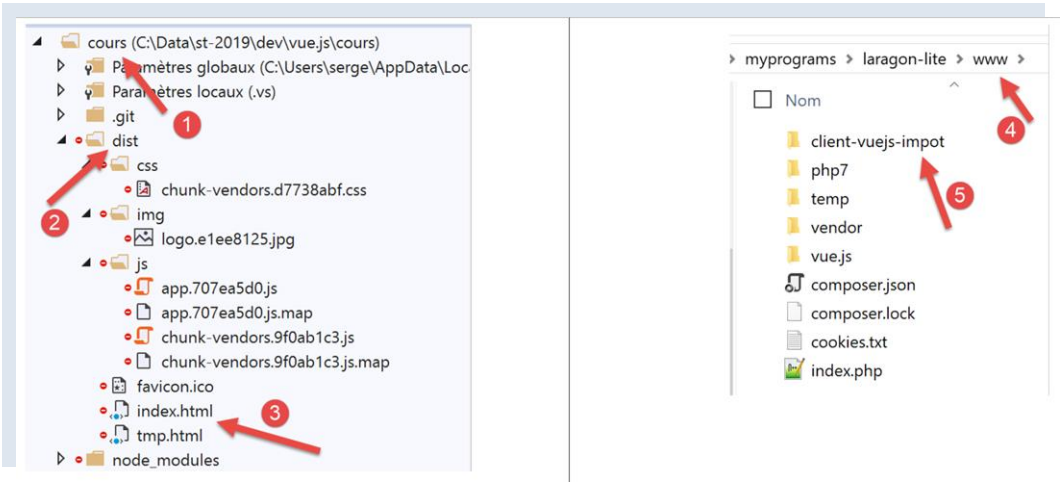
- en **[1-2]**, nous configurons la tâche **[build] [2]** dans le fichier **[package.json] [1]** ;
- en **[3-5]**, nous exécutons cette tâche. C'est elle qui va construire la version de production du projet **[vuejs-20]** ;

L'exécution de la tâche **[build]** se passe dans un terminal de **[VSCode]** :





- en **[3-6]**, des avertissements nous disent que le code généré est trop gros et qu'il faudrait le découper **[8]**. Cela relève de l'optimisation de l'architecture du code que nous n'aborderons pas ici ;
- en **[7]**, on nous dit que le dossier **[dist]** contient la version de production générée :



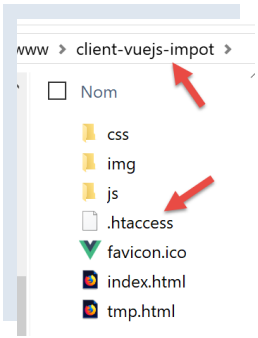
- en **[3]**, le fichier **[index.html]** est le fichier qui sera utilisé lorsqu'on demandera l'URL **[https://localhost:80/client-vuejs-impot/]**;

On a ici un site **statique** qui peut être déployé sur n'importe quel serveur. Nous allons le déployer sur le |serveur Laragon local|. Le dossier **[dist] [2]** est copié dans le dossier **[<laragon>/www] [4]** où <laragon> est le dossier d'installation du serveur Laragon. Nous renommons ce dossier **[client-vuejs-impot] [5]** puisque nous avons configuré la version de production pour fonctionner à l'URL **[/client-vuejs-impot/]**.

étape 3

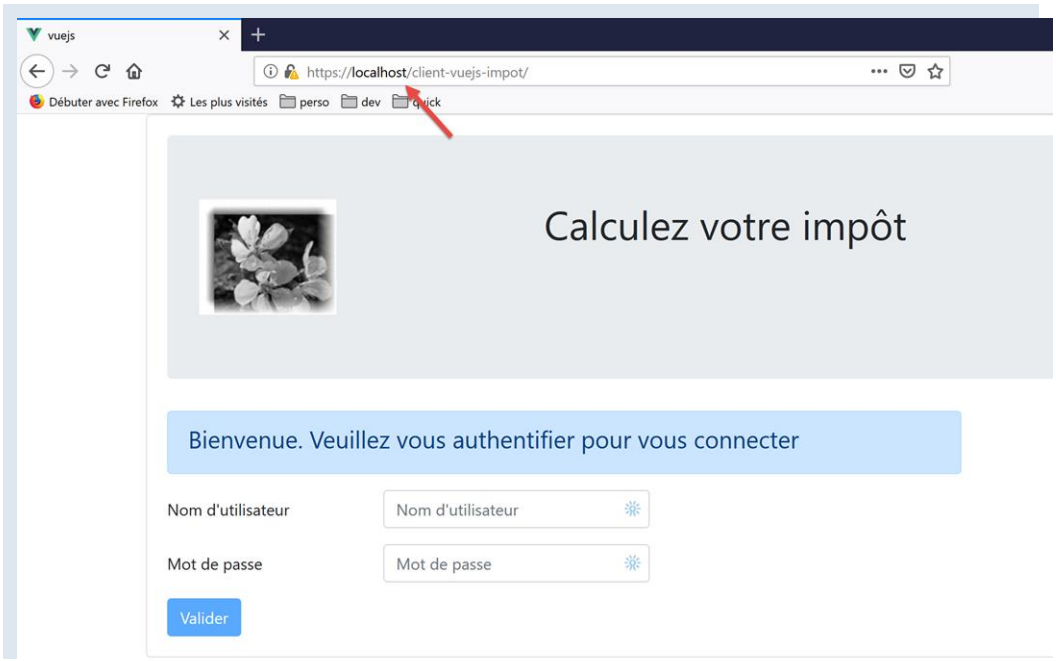
Nous ajoutons dans le dossier **[client-vuejs-impot]** qui vient d'être créé le fichier **[.htaccess]** suivant :

```
1.  <IfModule mod_rewrite.c>
2.    RewriteEngine On
3.    RewriteBase /client-vuejs-impot/
4.    RewriteRule ^index\.html$ - [L]
5.    RewriteCond %{REQUEST_FILENAME} !-f
6.    RewriteCond %{REQUEST_FILENAME} !-d
7.    RewriteRule . /client-vuejs-impot/index.html [L]
8.  </IfModule>
```



Ce fichier est un fichier de configuration du serveur web Apache. Si nous ne le mettons pas et que nous demandons directement l'URL **[https://localhost/client-vuejs-impot/calcul-impot]**, sans passer d'abord par l'URL **[https://localhost/client-vuejs-impot/]** nous obtenons une erreur 404. Avec ce fichier, nous obtenons bien la vue **[CalculImpot]**.

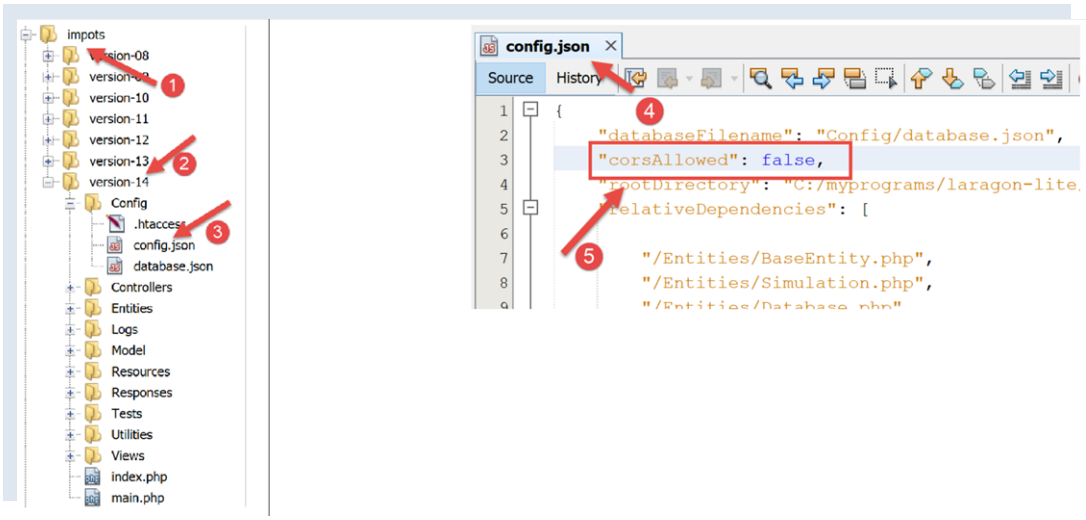
Ceci fait, nous lançons le serveur Laragon si ce n'est déjà fait et demandons l'URL **[https://localhost/client-vuejs-impot/]** :



Le lecteur est invité à tester la version de production de notre application.

Nous pouvons modifier le serveur de calcul de l'impôt sur un point : les entêtes CORS qu'il envoie systématiquement à ses clients. Cela avait été nécessité pour la version du client exécutée à partir du domaine **[localhost:8080]**. Maintenant que client et serveur s'exécutent tous deux dans le domaine **[localhost:80]**, les entêtes CORS deviennent inutiles.

Nous modifions le fichier **[config.json]** de la version 14 du serveur :

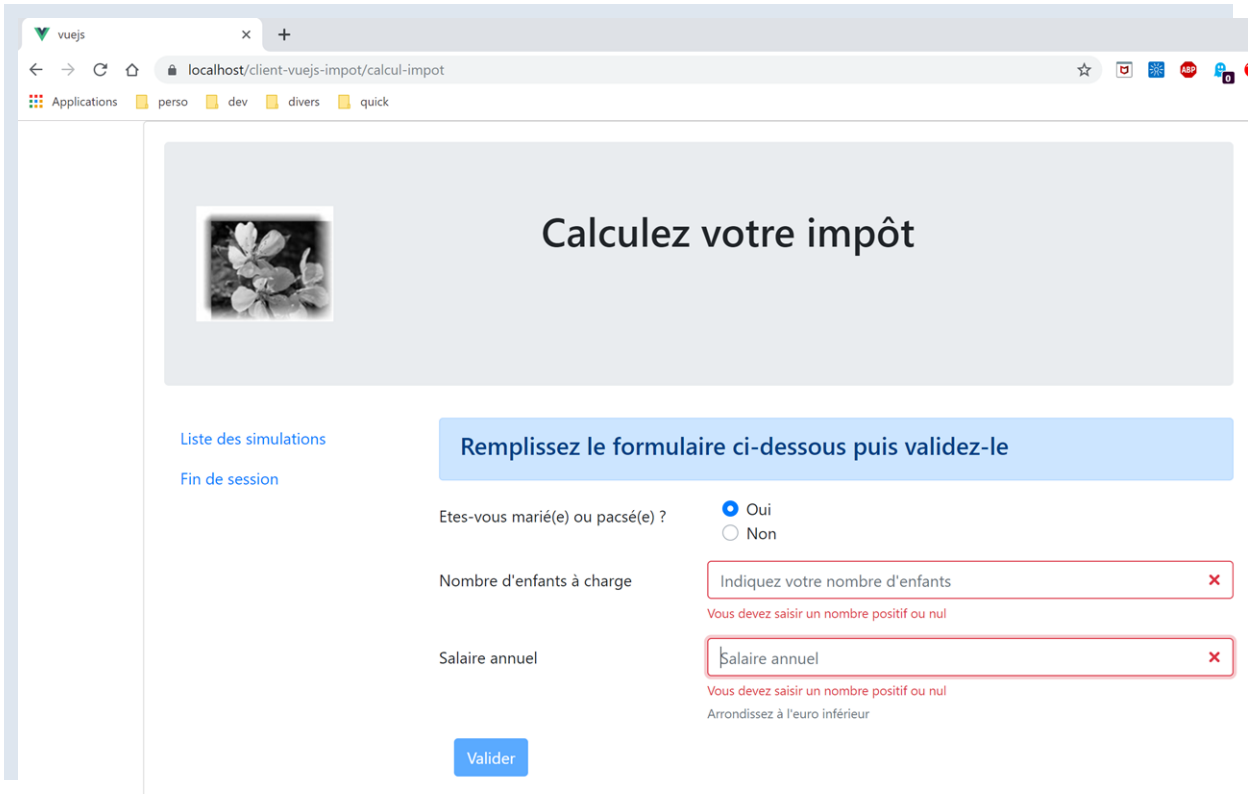


- en **[4]**, nous indiquons que désormais les requêtes CORS sont refusées ;

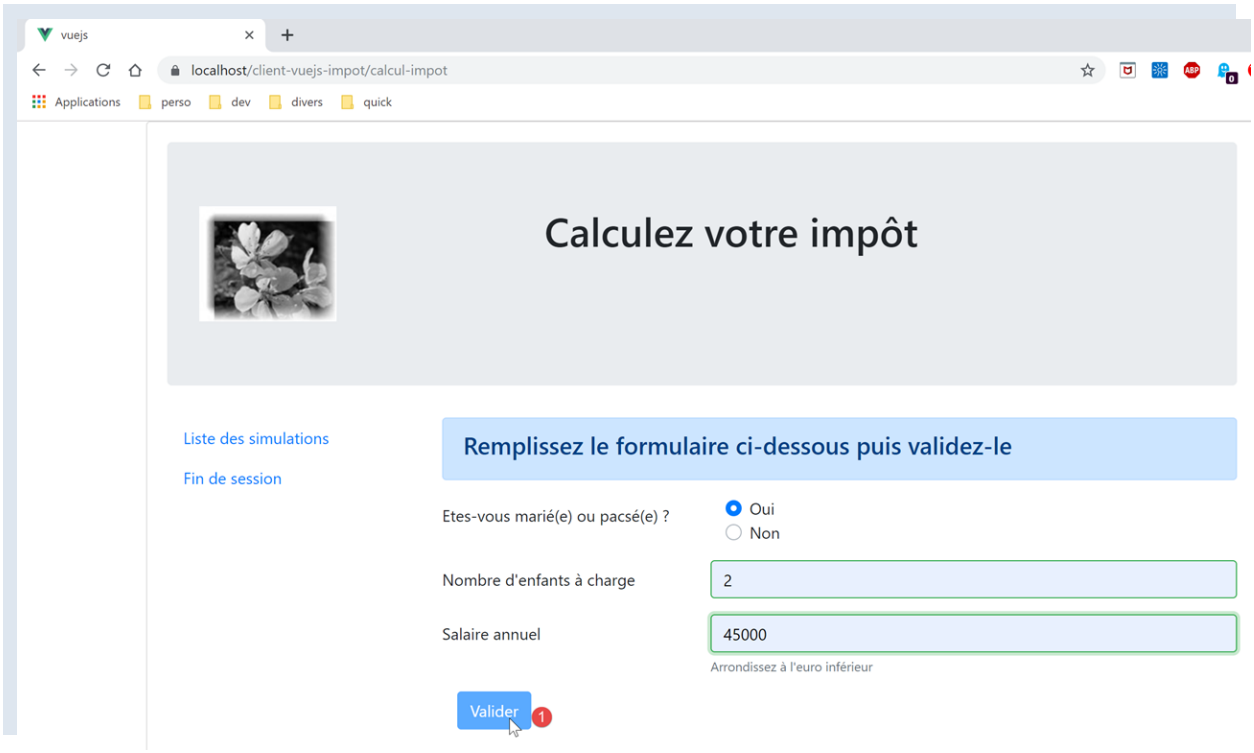
Sauvegardons cette modification et redemandons l'URL **[https://localhost/client-vuejs-impot/]**. Ca doit continuer à marcher.

3.18.7 Gestion des URL manuelles

Au lieu d'utiliser sagement les liens du menu de navigation, l'utilisateur peut vouloir taper les URL de l'application manuellement dans le champ d'adresse du navigateur. Demandons par exemple l'URL **[https://client-vuejs-impot/calcul-impot]** sans passer par la case d'authentification. Un hacker tenterait sûrement ça. On obtient la vue suivante :



On obtient bien la vue du calcul de l'impôt. Maintenant essayons de remplir les zones de saisie et de les valider :



On découvre alors que le bouton **[1] [Valider]** reste toujours désactivé même si les saisies sont correctes. Regardons le code de la vue **[FormCalculImpot]** :

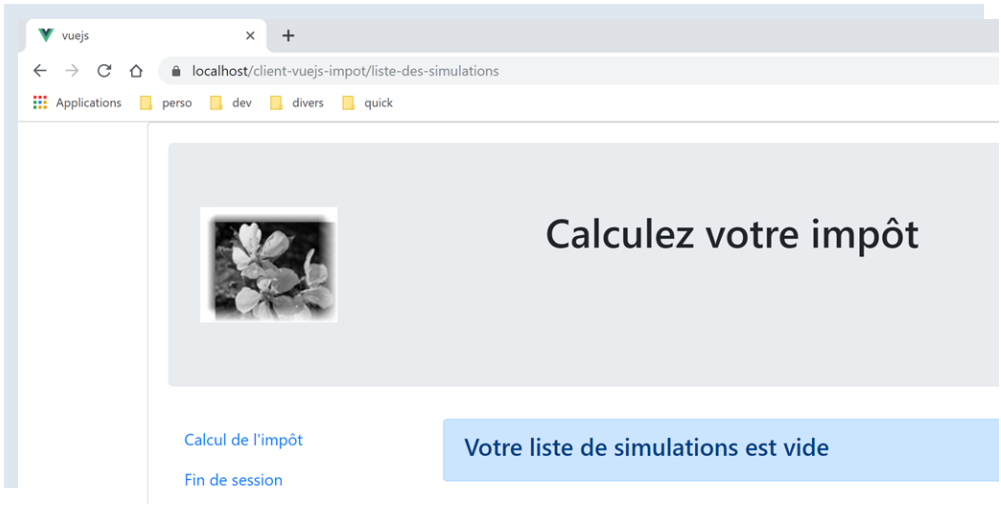
```
1.  <b-col cols="5">
2.        <b-button type="submit" variant="primary" :disabled="formInvalide">Valider</b-button>
3.  </b-col>
```

Ligne 2, on voit que son état actif / inactif dépend de la propriété **[formInvalide]**. Celle-ci est la propriété calculée suivante :

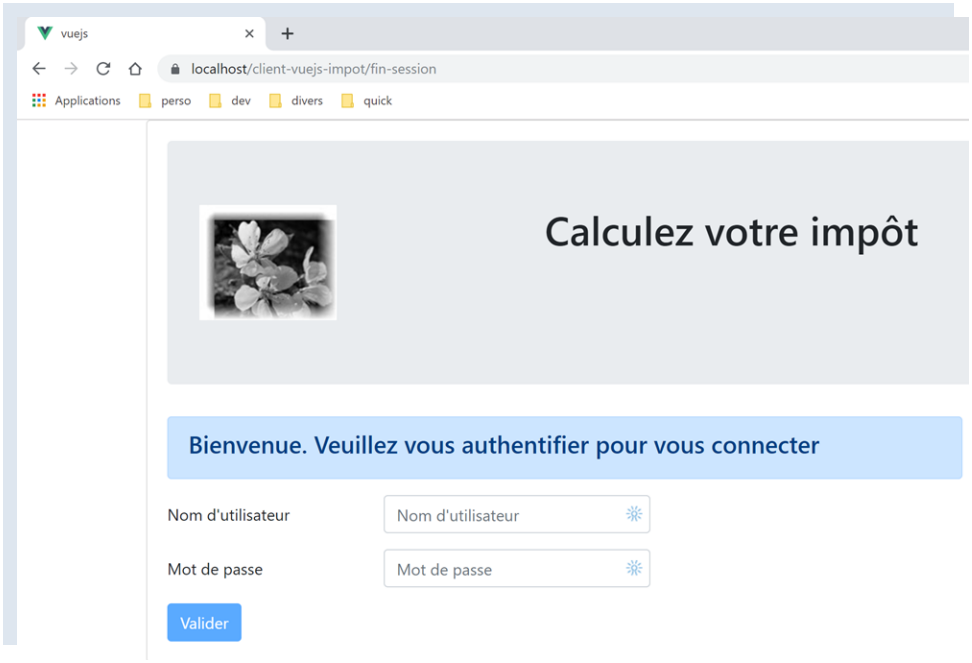
```
1.  formInvalide() {
2.        return (
3.          // salaire invalide
4.          !this.salaireValide ||
5.          // ou enfants invalide
6.          !this.enfantsValide ||
7.          // ou données fiscales pas obtenues
8.          !this.$métier.taxAdminData
9.        );
10. },
```

Ligne 8, on voit que pour que le formulaire soit valide, il faut avoir obtenu les données fiscales. Or celles-ci sont obtenues lors de la validation de la vue **[Authentification]** que l'utilisateur a 'sautée'. Il ne pourra donc pas valider le formulaire. S'il avait pu le faire, il aurait reçu un message d'erreur du serveur lui indiquant qu'il n'était pas authentifié. Les vérifications doivent toujours être faites côté serveur. Les vérifications côté navigateur peuvent toujours être contournées. Il suffit de prendre un client de type **[Postman]** qui enverra des requêtes brutes au serveur.

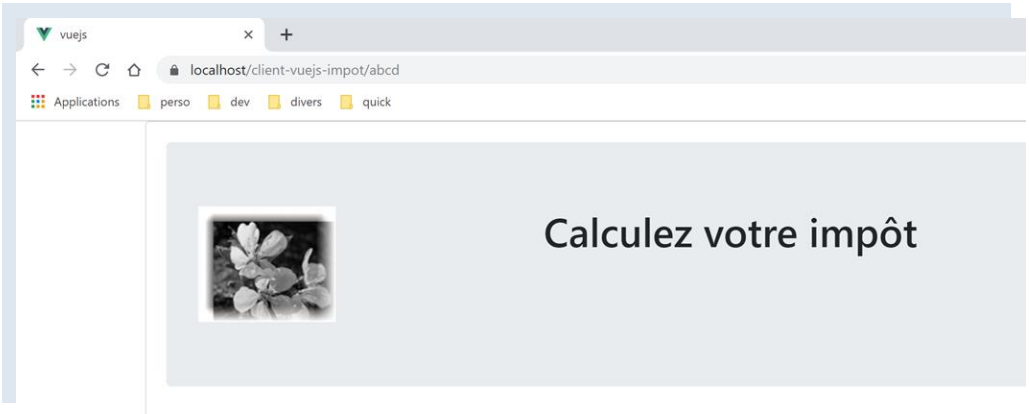
Maintenant demandons l'URL **[https://localhost/client-vuejs-impot/liste-des-simulations]**. On obtient la vue suivante :



Maintenant l'URL **[https://localhost/client-vuejs-impot/fin-session]**. Nous obtenons la vue suivante :

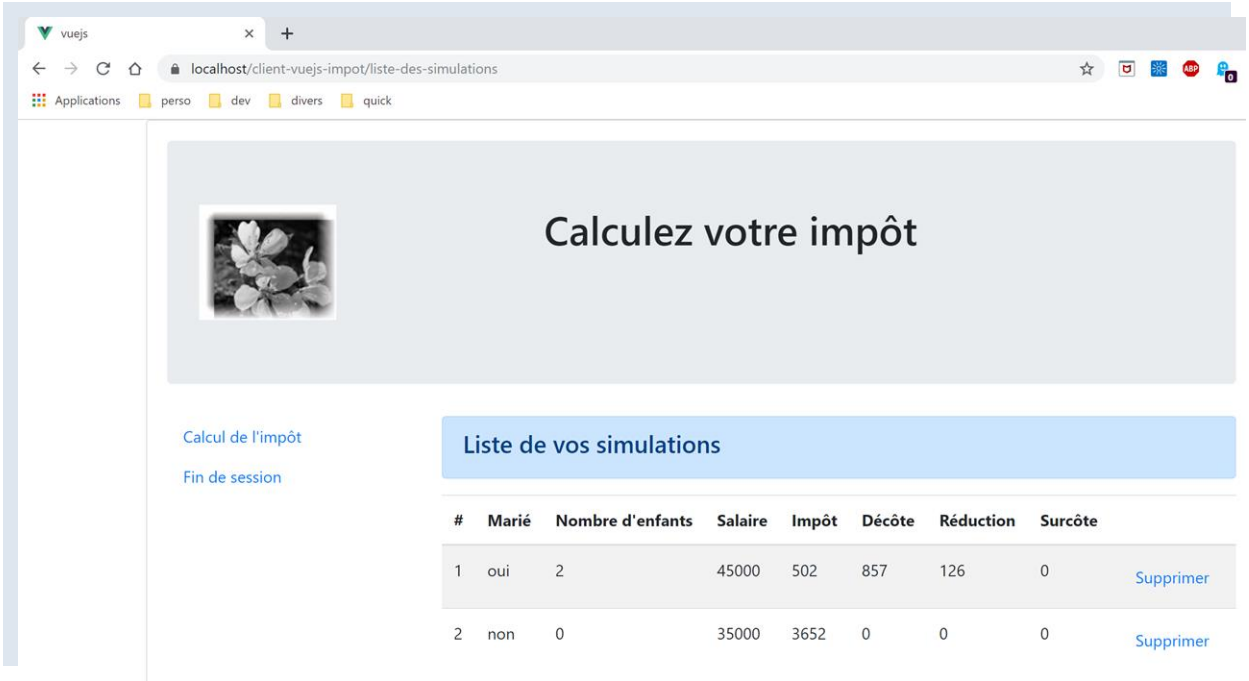


Maintenant une vue qui n'existe pas **[https://localhost/client-vuejs-impot/abcd]** :

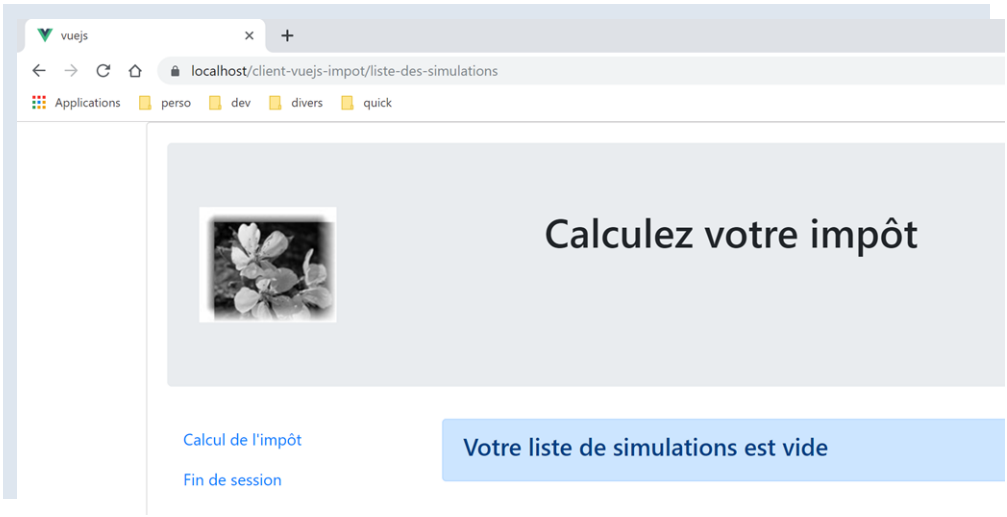


Notre application résiste plutôt bien aux URL tapées à la main. Lorsque celles-ci sont appelées, le routeur de l'application le sait. Il est donc possible d'intervenir avant que la vue ne soit finalement affichée. Nous allons regarder ce point dans le projet **[vuejs-21]**.

Un autre point à regarder est le suivant. Imaginons que l'utilisateur ait fait quelques simulations dans les règles :



Maintenant rafraîchissons la page par un F5 :



On a fait quelque chose de déconseillé : taper l'URL à la main (faire F5 revient à ça). Nous avons alors perdu nos simulations.

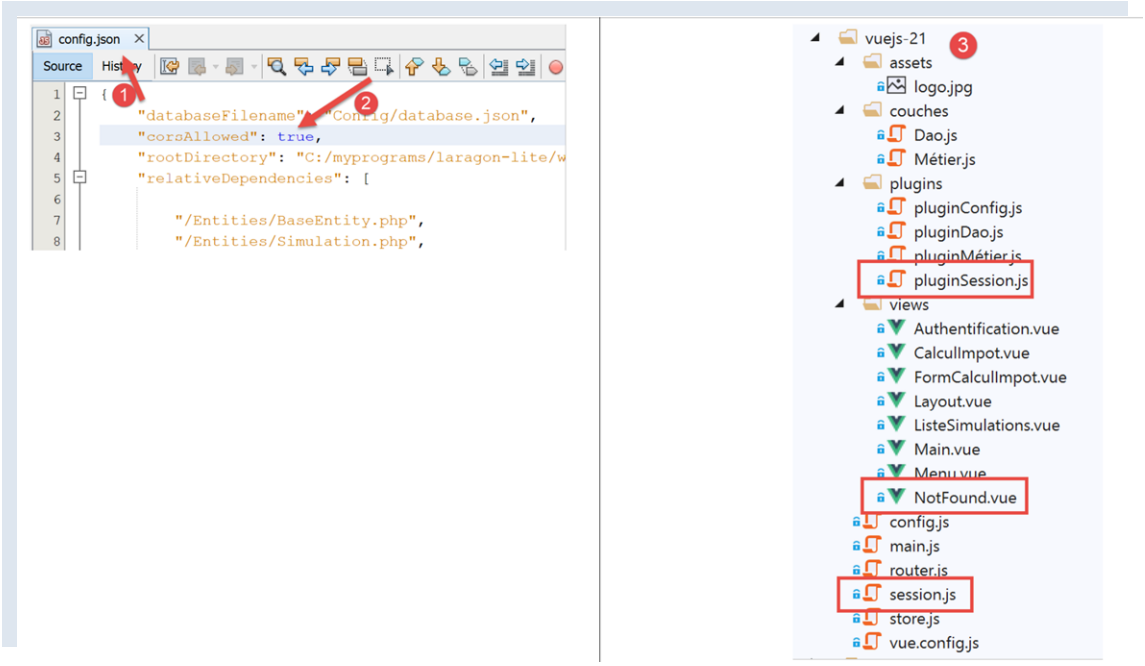
Le projet suivant **[vuejs-21]** se propose d'apporter deux améliorations :

- contrôler les URL tapées par l'utilisateur ;
- garder une mémoire de l'application même si l'utilisateur tape une URL. Ci-dessus, on voit qu'on a perdu la liste des simulations ;

3.19 Améliorations du client Vue.js

3.19.1 Introduction

Nous allons tester le projet **[vuejs-21]** avec le serveur de développement. Nous allons donc avoir besoin de nouveau que le serveur envoie les entêtes CORS. Il faut donc que le fichier **[config.json]** de la version 14 du serveur de calcul de l'impôt autorise ces entêtes :



Le projet **[vuejs-21]** est créé initialement par duplication du projet **[vuejs-20]**. Il est ensuite modifié **[3]**.

De nouveaux fichiers apparaissent :

- **[session.js]** : exporte un objet **[session]** qui va encapsuler des informations sur la session courante ;
- **[pluginSession]** : rend disponible l'objet **[session]** précédent dans la propriété **[$session]** des vues ;
- **[NotFound.vue]** : une nouvelle vue affichée lorsque l'utilisateur demande manuellement une URL qui n'existe pas ;

Des fichiers seront modifiés :

- **[main.js]** : va initialiser la session courante puis, lorsque l'utilisateur va taper manuellement des URL va la restaurer ;
- **[router.js]** : des contrôles sont ajoutés pour traiter le cas des URL tapées par l'utilisateur ;
- **[store.js]** : une nouvelle mutation est ajoutée ;
- **[config.js]** : une nouvelle configuration est ajoutée ;
- différentes vues essentiellement pour sauver la session courante à des moments clés de la vie de l'application. Celle-ci est ensuite restaurée à chaque fois que l'utilisateur tape manuellement des URL ;

3.19.2 Le store [Vuex]

Le script **[./store]** évolue de la façon suivante :

```
1.  // plugin Vuex
2.  import Vue from 'vue'
3.  import Vuex from 'vuex'
4.  Vue.use(Vuex);
5.
6.  // store Vuex
7.  const store = new Vuex.Store({
8.    state: {
9.      // le tableau des simulations
10.     simulations: [],
11.     // le n° de la dernière simulation
12.     idSimulation: 0
13.   },
14.   mutations: {
15.     // suppression ligne n° index
16.     deleteSimulation(state, index) {
17.       …
18.     },
19.     // ajout d'une simulation
20.     addSimulation(state, simulation) {
21.       …
22.     },
23.     // nettoyage state
24.     clear(state) {
```

```
25.       state.simulations = [];
26.       state.idSimulation = 1;
27.     }
28.   }
29. });
30. // export de l'objet [store]
31. export default store;
```

- lignes 24-29 : la mutation **[clear]** supprime la liste des simulations enregistrées et remet à 0 le n° de la dernière simulation.

3.19.3 La session

Le besoin d'une session vient du fait que lorsque l'utilisateur tape une URL dans le champ adresse du navigateur, le script **[main.js] est exécuté de nouveau**. Or celui-ci contient l'instruction :

```
1.  // store Vuex
2.  import store from './store'
```

Cette instruction importe le fichier **[./store]** suivant :

```
1.  // plugin Vuex
2.  import Vue from 'vue'
3.  import Vuex from 'vuex'
4.  Vue.use(Vuex);
5.
6.  // store Vuex
7.  const store = new Vuex.Store({
8.    state: {
9.      // le tableau des simulations
10.     simulations: [],
11.     // le n° de la dernière simulation
12.     idSimulation: 0
13.   },
14.   mutations: {
15.     ...
16.   }
17. });
18. // export de l'objet [store]
19. export default store;
```

On voit, lignes 7-13, qu'on importe un tableau de simulations vide. Si donc on avait des simulations avant que l'utilisateur ne tape une URL dans le champ adresse du navigateur, après on n'en a plus. L'idée est :

- d'utiliser une session qui stockerait les informations qu'on veut conserver si l'utilisateur tape manuellement des URL ;
- de la sauvegarder à des moments clés de l'application ;
- de la restaurer dans **[main.js]** qui **est toujours exécuté** lorsqu'une URL est tapée manuellement ;

Le script **[./session]** est le suivant :

```
1.  // on importe le store Vuex
2.  import store from './store'
3.  // on importe la configuration
4.  import config from './config';
5.
6.  // l'objet [session]
7.  const session = {
8.    // session démarrée
9.    started: false,
10.   // authentification
11.   authenticated: false,
12.   // heure de sauvegarde
13.   saveTime: "",
14.   // couche [métier]
15.   métier: null,
16.   // état Vuex
17.   state: null,
18.
19.   // sauvegarde de la session dans une chaîne jSON
20.   save() {
21.     // on ajoute à la session quelques proprités
22.     this.saveTime = Date.now();
23.     this.state = store.state;
24.     // on la transforme en jSON
```

```
25.     const json = JSON.stringify(this);
26.     // on la stocke sur le navigateur
27.     localStorage.setItem("session", json);
28.     // eslint-disable-next-line no-console
29.     console.log("session save", json);
30.   },
31.
32.   // restauration de la session
33.   restore() {
34.     // on récupère la session jSON à partir du navigateur
35.     const json = localStorage.getItem("session")
36.     // si on a récupéré qq chose
37.     if (json) {
38.       // on restaure toutes les clés de la session
39.       const restore = JSON.parse(json);
40.       for (var key in restore) {
41.         if (restore.hasOwnProperty(key)) {
42.           this[key] = restore[key];
43.         }
44.       }
45.       // si on a dépassé une certaine durée d'inactivité depuis le début de la session, on repart de zéro
46.       let durée = Date.now() - this.saveTime;
47.       if (durée > config.duréeSession) {
48.         // on vide la session - elle sera également sauvegardée
49.         session.clear();
50.       } else {
51.         // on régénère le store Vuex
52.         store.replaceState(JSON.parse(JSON.stringify(this.state)));
53.       }
54.     }
55.     // eslint-disable-next-line no-console
56.     console.log("session restore", this);
57.   },
58.
59.   // on nettoie la session
60.   clear() {
61.     // eslint-disable-next-line no-console
62.     console.log("session clear");
63.     // raz de certains champs de la session
64.     this.authenticated = false;
65.     this.saveTime = "";
66.     this.started = false;
67.     if (this.métier) {
68.       // on réinitialise le champ [taxAdminData]
69.       this.métier.taxAdminData = null;
70.     }
71.     // le store Vuex est nettoyé également
72.     store.commit("clear");
73.     // on sauvegarde la nouvelle session
74.     this.save();
75.   },
76. }
77.
78. // export de l'objet [session]
79. export default session;
```

Commentaires

- ligne 2 : la session va encapsuler également le store **[Vuex]** (liste des simulations, n° de la dernière simulation faite) ;
- lignes 7-17 : les informations conservées par la session :
  - **[started]** : la session jSON avec le serveur a démarré ou non ;
  - **[authenticated]** : l'utilisateur s'est authentifié ou pas ;
  - **[saveTime]** : la date en millisecondes de la dernière sauvegarde ;
  - **[métier]** : une référence sur la couche **[métier]**. Celle-ci contient la donnée **[taxAdminData]** qui permet le calcul de l'impôt ;
  - **[state]** : le state du store **[Vuex]** (liste des simulations, n° de la dernière simulation faite) ;
- lignes 20-30 : la méthode **[save]** sauvegarde la session localement sur le navigateur exécutant l'application ;
  - ligne 22 : on note l'heure de sauvegarde ;
  - ligne 23 : on récupère le **[state]** du store **[Vuex]** ;
  - ligne 25 : on crée la chaîne jSON de la session ;
  - ligne 27 : on la stocke localement sur le navigateur associée à la clé **[session]** ;
- lignes 33-57 : la méthode **[restore]** permet de restaurer une session à partir de sa sauvegarde locale sur le navigateur ;
  - ligne 35 : on récupère la sauvegarde jSON locale ;
  - ligne 37 : si on a récupéré quelque chose ;

- lignes 39-44 : l'objet **[session]** est reconstitué ;
- ligne 46 : on calcule la durée qui nous sépare de la dernière sauvegarde ;
- lignes 47-50 : si cette durée est supérieure à une valeur **[config.duréeSession]** fixée par configuration, la session est réinitialisée (ligne 49) et à cette occasion sauvegardée ;
- ligne 52 : sinon on régénère l'attribut **[state]** du store **[Vuex]** ;

- lignes 60-75 : la méthode **[clear]** réinitialise la session ;
  - lignes 64-70 : les propriétés de la session sont réinitialisées à leurs valeurs initiales ;
  - ligne 72 : ainsi que le store **[Vuex]** ;
  - ligne 74 : la nouvelle session est sauvegardée ;

3.19.4 Le fichier de configuration [config]

Le fichier **[./config]** évolue de la façon suivante :

```
1.  // utilisation de la bibliothèque [axios]
2.  const axios = require('axios');
3.  // timeout des requêtes HTTP
4.  axios.defaults.timeout = 2000;
5.  ...
6.
7.  // export de la configuration
8.  export default {
9.    // objet [axios]
10.   axios: axios,
11.   // délai maximal d'inactivité de la session : 5 mn = 300 s = 300000 ms
12.   duréeSession: 300000
13. }
```

- ligne 12 : on va gérer la session de l'application un peu comme on gère une session web. On fixe ici une durée d'inactivité maximale de 5 minutes ;

3.19.5 Le plugin [pluginSession]

Comme il a été fait déjà de nombreuses fois, le plugin **[pluginSession]** va permettre aux vues d'avoir accès à la session via la propriété **[this.$session]** :

```
1.  export default {
2.    install(Vue, session) {
3.      // ajoute une propriété [$session] à la classe vue
4.      Object.defineProperty(Vue.prototype, '$session', {
5.        // lorsque Vue.$session est référencé, on rend le 2ième paramètre [session]
6.        get: () => session,
7.      })
8.    }
9.  }
```

3.19.6 Le script principal [main]

Le script principal **[./main.js]** évolue de la façon suivante :

```
1.  // log de démarrage
2.  // eslint-disable-next-line no-console
3.  console.log("main started");
4.
5.  // imports
6.  import Vue from 'vue'
7.
8.  ...
9.
10. // instanciation couche [métier]
11. import Métier from './couches/Métier';
12. const métier = new Métier();
13.
14. // plugin [métier]
15. import pluginMétier from './plugins/pluginMétier'
16. Vue.use(pluginMétier, métier)
17.
18. // store Vuex
19. import store from './store'
20.
21. // session
22. import session from './session';
```

```
23. import pluginSession from './plugins/pluginSession'
24. Vue.use(pluginSession, session)
25.
26. // on restore la session avant de redémarrer
27. session.restore();
28.
29. // on restaure la couche [métier]
30. if (session.métier && session.métier.taxAdminData) {
31.   métier.setTaxAdminData(session.métier.taxAdminData);
32. }
33.
34. // démarrage de l'UI
35. new Vue({
36.   el: '#app',
37.   // le routeur
38.   router: router,
39.   // le store Vuex
40.   store: store,
41.   // la vue principale
42.   render: h => h(Main),
43. })
44.
45. // log de fin
46. // eslint-disable-next-line no-console
47. console.log("main terminated, session=", session);
```

- ligne 19 : on importe la session ;
- ligne 20 : on importe son plugin ;
- ligne 21 : le plugin **[pluginSession]** est intégré à **[Vue]**. Après cette instruction toutes les vues disposent de la session dans leur attribut **[$session]** ;
- ligne 27 : la session est restaurée. La session importée ligne 11 est alors initialisée avec le contenu de sa dernière sauvegarde ;
- après la ligne 16, les vues disposent d'une propriété **[$métier]** initialisée ligne 12. Cette propriété n'a pas l'information **[taxAdminData]** qui permet de calculer l'impôt ;
- lignes 30-32 : si la restauration qui vient d'être faite a restauré la propriété **[session.métier.taxAdminData]** alors la propriété **[$métier]** des vues est initialisée avec cette valeur ;

3.19.7 Le fichier de routage [router]

Le fichier de routage **[./router]** évolue comme suit :

```
1.  // imports
2.  import Vue from 'vue'
3.  import VueRouter from 'vue-router'
4.  // les vues
5.  import Authentification from './views/Authentification'
6.  import CalculImpot from './views/CalculImpot'
7.  import ListeSimulations from './views/ListeSimulations'
8.  import NotFound from './views/NotFound'
9.  // la session
10. import session from './session'
11.
12. // plugin de routage
13. Vue.use(VueRouter)
14.
15. // les routes de l'application
16. const routes = [
17.   // authentification
18.   { path: '/', name: 'authentification', component: Authentification },
19.   { path: '/authentification', name: 'authentification2', component: Authentification },
20.   // calcul de l'impôt
21.   {
22.     path: '/calcul-impot', name: 'calculImpot', component: CalculImpot,
23.     meta: { authenticated: true }
24.   },
25.   // liste des simulations
26.   {
27.     path: '/liste-des-simulations', name: 'listeSimulations', component: ListeSimulations,
28.     meta: { authenticated: true }
29.   },
30.   // fin de session
31.   {
32.     path: '/fin-session', name: 'finSession'
33.   },
34.   // page inconnue
```

```
35.   {
36.     path: '*', name: 'notFound', component: NotFound,
37.   },
38. ]
39.
40. // le routeur
41. const router = new VueRouter({
42.   // les routes
43.   routes,
44.   // le mode d'affichage des URL
45.   mode: 'history',
46.   // l'URL de base de l'application
47.   base: '/client-vuejs-impot/'
48. })
49.
50. // vérification des routes
51. router.beforeEach((to, from, next) => {
52.   // eslint-disable-next-line no-console
53.   console.log("router to=", to, "from=", from);
54.   // route réservée aux utilisateurs authentifiés ?
55.   if (to.meta.authenticated && !session.authenticated) {
56.     next({
57.       // on passe à l'authentification
58.       name: 'authentification',
59.     })
60.     // retour à la boucle événementielle
61.     return;
62.   }
63.   // cas particulier de la fin de session
64.   if (to.name === "finSession") {
65.     // on nettoie la session
66.     session.clear();
67.     // on va sur la vue [authentification]
68.     next({
69.       name: 'authentification',
70.     })
71.     // retour à la boucle événementielle
72.     return;
73.   }
74.   // autres cas - vue suivante normale du routage
75.   next();
76. })
77.
78. // export du router
79. export default router
```

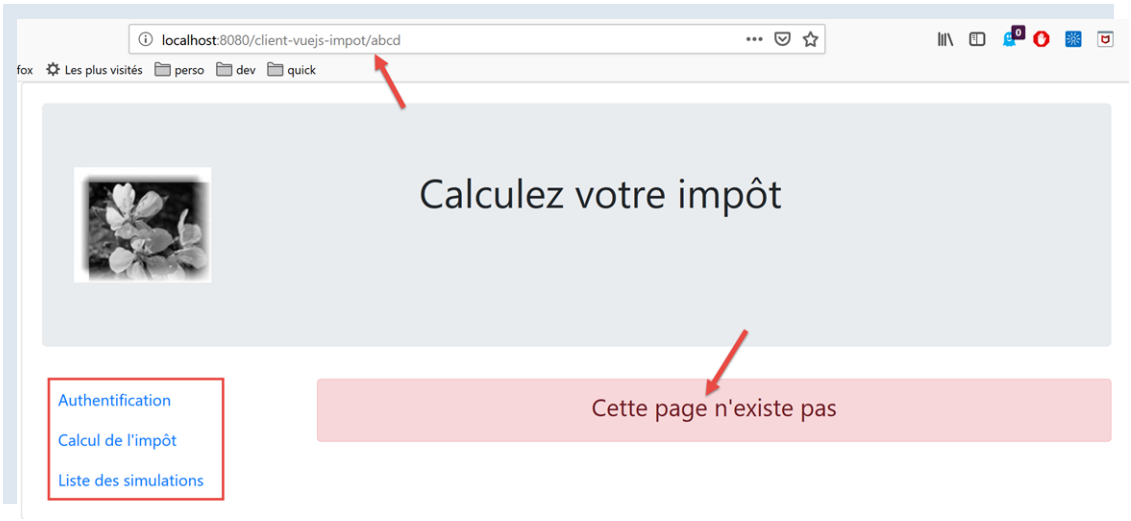
Commentaires

- lignes 16-38 : certaines routes ont été enrichies d'informations supplémentaires ;
- ligne 19 : on a créé une nouvelle route pour aller à la vue **[Authentification]** ;
- lignes 21-24 : la route qui mène à la vue **[CalculImpot]** a maintenant une propriété **[meta]** (ce nom est obligatoire). Le contenu de cet objet peut être quelconque et est fixé par le développeur ;
- ligne 23 : on met dans **[meta]**, la propriété **[authenticated]** (ce nom peut être quelconque). Il signifiera pour nous que pour aller à la vue **[CalculImpot]**, l'utilisateur doit être authentifié ;
- lignes 26-29 : on fait la même chose pour la route qui mène à la vue **[ListeSimulations]**. Là aussi, l'utilisateur doit être authentifié ;
- la propriété **[meta.authenticated]** va nous permettre de vérifier qu'un utilisateur qui tape manuellement les URL des vues **[CalculImpot, ListeSimulations]** ne peut pas les obtenir s'il n'est pas authentifié ;
- lignes 51-76 : la méthode **[beforeEach]** est exécutée avant qu'une vue ne soit routée. C'est le bon moment pour faire des vérifications ;
  - **[to]** : la prochaine route si on ne fait rien ;
  - **[from]** : la dernière route affichée ;
  - **[next]** : fonction permettant de changer la prochaine route affichée ;
- ligne 55 : on regarde si la prochaine route demande à ce que l'utilisateur soit authentifié ;
- lignes 56-59 : si oui et que l'utilisateur n'est pas authentifié, on change la prochaine route vers la vue **[Authentification]** ;
- lignes 64-73 : on traite le cas particulier de la route **[finSession]** des lignes 30-32. Celle-ci n'a pas de vue associée ;
  - ligne 66 : on réinitialise la session à sa valeur initiale ;
  - lignes 68-70 : on programme la vue **[Authentification]** comme prochaine vue ;
- ligne 75 : si on n'est pas dans les deux cas précédents, on se contente de passer à la route prévue par le fichier de routage ;

- lignes 35-37 : on prévoit une vue **[NotFound]** si la route tapée par l'utilisateur ne correspond à aucune route connue. Cette vue est importée ligne 8. Les routes sont vérifiées dans l'ordre du fichier de routage. Si donc on arrive à la ligne 36, c'est que la route demandée n'est aucune des routes des lignes 18-33 ;

3.19.8 La vue [NotFound]

La vue **[NotFound]** est affichée si la route tapée par l'utilisateur ne correspond à aucune route connue :



Le code de la vue est le suivant :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- mise en page -->
4.    <Layout :left="true" :right="true">
5.      <!-- alerte dans la colonne de droite -->
6.      <template slot="right">
7.        <!-- message sur fond jaune -->
8.        <b-alert show variant="danger" align="center">
9.          <h4>Cette page n'existe pas</h4>
10.       </b-alert>
11.     </template>
12.     <!-- menu de navigation dans la colonne de gauche -->
13.     <Menu slot="left" :options="options" />
14.   </Layout>
15. </template>
16.
17. <script>
18. // imports
19. import Layout from "./Layout";
20. import Menu from "./Menu";
21. export default {
22.   // composants
23.   components: {
24.     Layout,
25.     Menu
26.   },
27.   // état interne du composant
28.   data() {
29.     return {
30.       // options du menu de navigation
31.       options: [
32.         {
33.           text: "Authentification",
34.           path: "/"
35.         }
36.       ]
37.     };
38.   },
39.   // cycle de vie
40.   created() {
41.     // eslint-disable-next-line
42.     console.log("NotFound created");
43.     // on regarde quelles options de menu offrir
```

```
44.     if (this.$session.authenticated && this.$métier.taxAdminData) {
45.       // l'utilisateur peut faire des simulations
46.       Array.prototype.push.apply(this.options, [
47.         {
48.           text: "Calcul de l'impôt",
49.           path: "/calcul-impot"
50.         },
51.         {
52.           text: "Liste des simulations",
53.           path: "/liste-des-simulations"
54.         }
55.       ]);
56.     }
57.   }
58. };
59. </script>
```

Commentaires

- ligne 4 : elle utilise les deux colonnes des vues routées ;
- lignes 6-11 : un message d'erreur ;
- ligne 13 : le menu de navigation occupe la colonne de gauche ;
- lignes 31-36 : les options par défaut du menu ;
- lignes 40-57 : code exécuté lorsque la vue est créée ;
- ligne 44 : on regarde si l'utilisateur peut faire des simulations ;
- lignes 45-55 : si oui, on ajoute deux options au menu de navigation, celles où il faut être authentifié et avoir une couche **[métier]** opérationnelle (lignes 46-55) ;

3.19.9 La vue [Authentification]

La vue **[Authentification]** évolue comme suit :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <Layout :left="false" :right="true">
4.      …
5.    </Layout>
6.  </template>
7.
8.  <!-- dynamique de la vue -->
9.  <script>
10. import Layout from "./Layout";
11. export default {
12.   // état du composant
13.   data() {
14.     return {
15.       // utilisateur
16.       user: "",
17.       // son mot de passe
18.       password: "",
19.       // contrôle l'affichage d'un msg d'erreur
20.       showError: false,
21.       // le message d'erreur
22.       message: ""
23.     };
24.   },
25.
26.   // composants utilisés
27.   components: {
28.     Layout
29.   },
30.
31.   // propriétés calculées
32.   computed: {
33.     // saisies valides
34.     valid() {
35.       return this.user && this.password && this.$session.started;
36.     }
37.   },
38.
39.   // gestionnaires d'évts
40.   methods: {
41.     // ---------- authentification
42.     async login() {
```

```
      try {
        // début attente
        this.$emit("loading", true);
        // on n'est pas encore authentifié
        this.$session.authenticated = false;
        // authentification bloquante auprès du serveur
        const response = await this.$dao.authentifierUtilisateur(
          this.user,
          this.password
        );
        // fin du chargement
        this.$emit("loading", false);
        // analyse de la réponse du serveur
        if (response.état != 200) {
          // on affiche l'erreur
          this.message = response.réponse;
          this.showError = true;
          // retour à la boucle événementielle
          return;
        }
        // pas d'erreur
        this.showError = false;
        // on est authentifié
        this.$session.authenticated = true;
        // --------- on demande maintenant les données de l'administration fiscale
        // au départ, pas de donnée
        this.$métier.setTaxAdminData(null);
        // début attente
        this.$emit("loading", true);
        // demande bloquante auprès du serveur
        const response2 = await this.$dao.getAdminData();
        // fin du chargement
        this.$emit("loading", false);
        // analyse de la réponse
        if (response2.état != 1000) {
          // on affiche l'erreur
          this.message = response2.réponse;
          this.showError = true;
          // retour à la boucle événementielle
          return;
        }
        // pas d'erreur
        this.showError = false;
        // on mémorise dans la couche [métier] la donnée reçue
        this.$métier.setTaxAdminData(response2.réponse);
        // on peut passer au calcul de l'impôt
        this.$router.push({ name: "calculImpot" });
      } catch (error) {
        // on remonte l'erreur au composant principal
        this.$emit("error", error);
      } finally {
        // maj session
        this.$session.métier = this.$métier;
        // on sauvegarde la session
        this.$session.save();
      }
    }
  },
  // cycle de vie : le composant vient d'être créé
  created() {
    // eslint-disable-next-line
    console.log("Authentification created");
    // l'utilisateur peut-il faire des simulations ?
    if (
      this.$session.started &&
      this.$session.authenticated &&
      this.$métier.taxAdminData
    ) {
      // alors l'utilisateur peut faire des simulations
      this.$router.push({ name: "calculImpot" });
      // retour à la boucle événementielle
      return;
    }
    // si la session jSON a déjà été démarrée, on ne la redémarre pas de nouveau
    if (!this.$session.started) {
      // début attente
      this.$emit("loading", true);
```

```
120.      // on initialise la session avec le serveur - requête asynchrone
121.      // on utilise la promesse rendue par les méthodes de la couche [dao]
122.      this.$dao
123.        // on initialise une session jSON
124.        .initSession()
125.        // on a obtenu la réponse
126.        .then(response => {
127.          // fin attente
128.          this.$emit("loading", false);
129.          // analyse de la réponse
130.          if (response.état != 700) {
131.            // on affiche l'erreur
132.            this.message = response.réponse;
133.            this.showError = true;
134.            // retour à la boucle événementielle
135.            return;
136.          }
137.          // la session a démarré
138.          this.$session.started = true;
139.        })
140.        // en cas d'erreur
141.        .catch(error => {
142.          // on remonte l'erreur à la vue [Main]
143.          this.$emit("error", error);
144.        })
145.        // dans tous les cas
146.        .finally(() => {
147.          // on sauvegarde la session
148.          this.$session.save();
149.        });
150.    }
151.  }
152.};
153.</script>
```

Commentaires

- on a surligné en jaune les instructions qui utilisent la session introduite dans cette version du client **[Vue.js]** ;
- lignes 97, 148 : à la fin des méthodes **[login, created]**, la session est sauvegardée quelque soit le résultat des requêtes HTTP qui ont lieu dans ces méthodes (clause **[finally]** dans les deux cas) ;
- la méthode **[created]** des lignes 102-150 est exécutée à chaque fois que la vue **[Authentification]** est créée. Si c'est l'utilisateur qui a tapé l'URL de la vue, la session va nous permettre de savoir quoi faire ;
- lignes 106-115 : si la session jSON est démarrée, l'utilisateur authentifié et la donnée **[this.$métier.taxAdminData]** initialisée alors l'utilisateur peut directement aller au formulaire de calcul de l'impôt (ligne 112) ;
- ligne 117 : la méthode **[created]** était utilisée dans la version précédente pour initialiser une session jSON avec le serveur. Cette phase est inutile si elle a déjà eu lieu ;
- lignes 42-66 : la méthode d'authentification ;
- ligne 66 : si l'authentification réussit, on le note dans la session ;
- lignes 67-92 : la demande au serveur des données de l'administration fiscale **[taxAdminData]** ;
- ligne 95 : à la fin de cette phase, on met à jour la propriété **[métier]** de la session que l'opération ait réussi ou pas ;

3.19.10 La vue [CalculImpot]

Le code de la vue **[CalculImpot]** évolue comme suit :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    …
4.  </template>
5.
6.  <script>
7.  // imports
8.  import FormCalculImpot from "./FormCalculImpot";
9.  import Menu from "./Menu";
10. import Layout from "./Layout";
11.
12. export default {
13.   // état interne
14.   data() {
15.     return {
16.       // options du menu
17.       options: [
```

```
18.         {
19.           text: "Liste des simulations",
20.           path: "/liste-des-simulations"
21.         },
22.         {
23.           text: "Fin de session",
24.           path: "/fin-session"
25.         }
26.       ],
27.       // résultat du calcul de l'impôt
28.       résultat: "",
29.       résultatObtenu: false
30.     };
31.   },
32.   // composants utilisés
33.   components: {
34.     Layout,
35.     FormCalculImpot,
36.     Menu
37.   },
38.   // méthodes de gestion des évts
39.   methods: {
40.     // résultat du calcul de l'impôt
41.     handleResultatObtenu(résultat) {
42.       // on construit le résultat en chaîne HTML
43.       …
44.       // une simulation de +
45.       this.$store.commit("addSimulation", résultat);
46.       // on sauvegarde la session
47.       this.$session.save();
48.     }
49.   },
50.   // cycle de vie
51.   created() {
52.     // eslint-disable-next-line
53.     console.log("CalculImpot created");
54.   }
55. };
56. </script>
```

Commentaires

- ligne 45 : la simulation calculée est ajoutée au store **[Vuex]**. Cela a un impact sur la session qui englobe la propriété **[state]** du store. Aussi sauvegarde-t-on la session (ligne 47) ;
- ligne 51 : on crée une méthode **[created]** pour suivre dans les logs les créations des vues ;

3.19.11 La vue [ListeSimulations]

La vue **[ListeSimulations]** évolue comme suit :

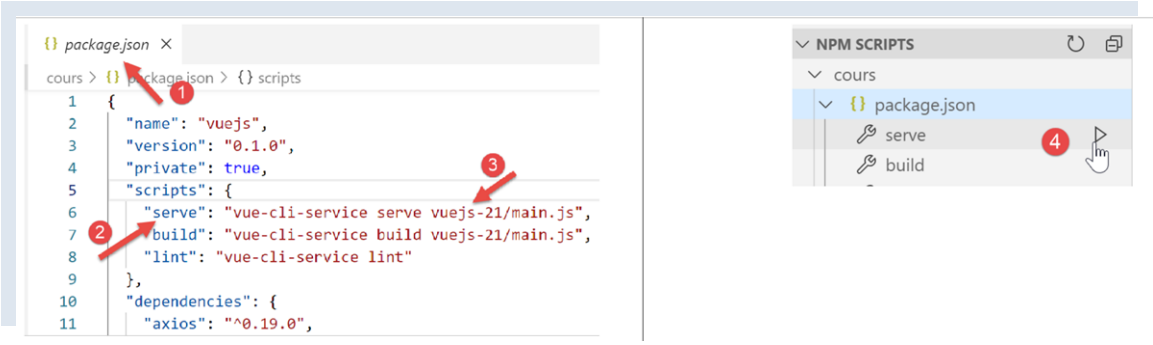
```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    …
4.  </template>
5.
6.  <script>
7.  // imports
8.  import Layout from "./Layout";
9.  import Menu from "./Menu";
10. export default {
11.   // composants
12.   components: {
13.     Layout,
14.     Menu
15.   },
16.   // état interne
17.   data() {
18.     …
19.   },
20.   // état interne calculé
21.   computed: {
22.     // liste des simulations prise dans le store Vuex
23.     simulations() {
24.       return this.$store.state.simulations;
25.     }
```

```
26.   },
27.   // méthodes
28.   methods: {
29.     supprimerSimulation(index) {
30.       // eslint-disable-next-line
31.       console.log("supprimerSimulation", index);
32.       // suppression de la simulation n° [index]
33.       this.$store.commit("deleteSimulation", index);
34.       // on sauvegarde la session
35.       this.$session.save();
36.     }
37.   },
38.   // cycle de vie
39.   created() {
40.     // eslint-disable-next-line
41.     console.log("ListeSimulations created");
42.   }
43. };
44. </script>
```

Commentaires

- ligne 35 : après la suppression d'une simulation ligne 34, on sauvegarde la session pour tenir compte de ce changement d'état ;
- lignes 39-42 : on continue à suivre la création des vues ;

3.19.12 Exécution du projet



Lors des tests vérifiez les points suivants :

- si l'utilisateur 'utilise' l'application via les liens du menu de navigation et les boutons / liens d'action, celle-ci fonctionne ;
- si l'utilisateur tape manuellement des URL, l'application continue à fonctionner. Faites en particulier le test suivant :
  - faites simulations ;
  - une fois sur la vue **[ListeSimulations]**, rechargez (F5) la vue. Dans l'application précédente **[vuejs-20]**, on perdait alors les simulations. Ici ce n'est pas le cas : on retrouve bien les simulations déjà faites ;
- regardez les logs pour comprendre :
  - à quel moment le script **[main]** est exécuté. Vous devez voir qu'il l'est à chaque fois que l'utilisateur tape une URL à la main ;
  - à quels moments les vues sont créées. Vous devez voir qu'elles le sont à chaque fois qu'elles vont être affichées ;
  - le fonctionnement du routage. Avant chaque routage un log est fait qui vous indique :
    - la route d'où vous venez ;
    - la route où vous allez ;

3.19.13 Déploiement de l'application sur un serveur local

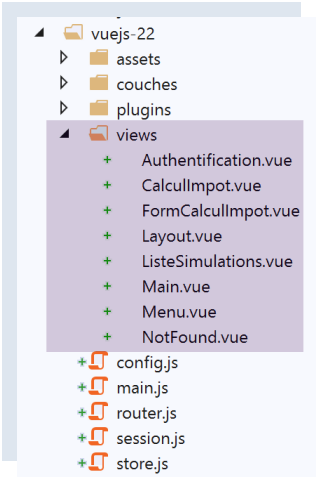
Comme exercice, suivez le paragraphe |Déploiement sur un serveur local|, pour déployer le projet **[vuejs-21]** sur le serveur Laragon local. Puis testez-le.

3.19.14 Mise au point de la version mobile

Théoriquement, l'utilisation de Bootstrap devrait nous permettre d'avoir une application utilisable sur différents média : smartphone, tablette, ordinateurs portable et de bureau. Cequi différencie ces média c'est la taille de leur écran.

Si on teste la version **[vuejs-21]** sur un mobile, on constate que c'est le chaos dans l'affichage des vues. La version **[vuejs-22]** corrige ce point. Les modifications ont toutes lieu dans les templates des vues. Elles ont consisté essentiellement à mettre au point un affichage

pour un écran de smartphone. Lorsque celui-ci est au point, l'affichage sur des écrans de taille plus importante se passe de façon fluide grâce à Bootstrap.



3.19.14.1 La vue `[Main]`

La vue **`[Main]`** évolue de la façon suivante :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div class="container">
4.      <b-card>
5.        <!-- jumbotron -->
6.        <b-jumbotron>
7.          <b-row>
8.            <b-col sm="4">
9.              <img src="../assets/logo.jpg" alt="Cerisier en fleurs" />
10.           </b-col>
11.           <b-col sm="8">
12.             <h1>Calculez votre impôt</h1>
13.           </b-col>
14.         </b-row>
15.       </b-jumbotron>
16.       ....
17.     </b-card>
18.   </div>
19. </template>
```

Commentaires

- ligne 8 : là où il y avait **`[cols='4']`** on écrit **`[sm='4']`**. **`[sm]`** signifie **`[small]`**. Les écrans des smartphones tombent dans cette catégorie. Les autres catégories sont **`[xs=extra small, md=medium, lg=large, xl=extra large]`** ;
- ligne 11 : idem ;

3.19.14.2 La vue `[Layout]`

La vue **`[Layout]`** évolue comme suit :

```
1.  <!-- définition HTML de la mise en page de la vue routée -->
2.  <template>
3.    <!-- ligne -->
4.    <div>
5.      <b-row>
6.        <!-- zone de trois colonnes à gauche -->
7.        <b-col sm="3" v-if="left">
8.          <slot name="left" />
9.        </b-col>
10.       <!-- zone de neuf colonnes à droite -->
11.       <b-col sm="9" v-if="right">
12.         <slot name="right" />
13.       </b-col>
14.     </b-row>
15.   </div>
16. </template>
```

3.19.14.3 La vue [Authentification]

La vue **[Authentification]** évolue comme suit :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <Layout :left="false" :right="true">
4.      <template slot="right">
5.        <!-- formulaire HTML - on poste ses valeurs avec l'action [authentifier-utilisateur] -->
6.        <b-form @submit.prevent="login">
7.          <!-- titre -->
8.          <b-alert show variant="primary">
9.            <h4>Bienvenue. Veuillez vous authentifier pour vous connecter</h4>
10.         </b-alert>
11.         <!-- 1ère ligne -->
12.         <b-form-group label="Nom d'utilisateur" label-for="user" description="Tapez admin">
13.           <!-- zone de saisie user -->
14.           <b-col sm="6">
15.             <b-form-input type="text" id="user" placeholder="Nom d'utilisateur" v-model="user" />
16.           </b-col>
17.         </b-form-group>
18.         <!-- 2ième ligne -->
19.         <b-form-group label="Mot de passe" label-for="password" description="Tapez admin">
20.           <!-- zone de saisie password -->
21.           <b-col sm="6">
22.             <b-input type="password" id="password" placeholder="Mot de passe" v-model="password" />
23.           </b-col>
24.         </b-form-group>
25.         <!-- 3ième ligne -->
26.         <b-alert
27.           show
28.           variant="danger"
29.           v-if="showError"
30.           class="mt-3"
31.         >L'erreur suivante s'est produite : {{message}}</b-alert>
32.         <!-- bouton de type [submit] sur une 3ième ligne -->
33.         <b-row>
34.           <b-col sm="2">
35.             <b-button variant="primary" type="submit" :disabled="!valid">Valider</b-button>
36.           </b-col>
37.         </b-row>
38.       </b-form>
39.     </template>
40.   </Layout>
41. </template>
```

Commentaires

- lignes 11 et 19 : on a supprimé l'attribut **[label-cols]** qui fixait un nombre de colonnes au label de la saisie. En l'absence de cet attribut, le label est au-dessus de la zone de saisie. Cela convient mieux aux écrans des smartphones ;

3.19.14.4 La vue [CalculImpot]

La vue **[CalculImpot]** évolue comme suit :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <Layout :left="true" :right="true">
5.        <!-- formulaire de calcul de l'impôt à droite -->
6.        <FormCalculImpot slot="right" @resultatObtenu="handleResultatObtenu" />
7.        <!-- menu de navigation à gauche -->
8.        <Menu slot="left" :options="options" />
9.      </Layout>
10.     <!-- zone d'affichage des résultat du calcul de l'impôt sous le formulaire -->
11.     <b-row v-if="résultatObtenu" class="mt-3">
12.       <!-- zone de trois colonnes vide -->
13.       <b-col sm="3" />
14.       <!-- zone de neuf colonnes -->
15.       <b-col sm="9">
16.         <b-alert show variant="success">
17.           <span v-html="résultat"></span>
18.         </b-alert>
19.       </b-col>
20.     </b-row>
```

```
21.   </div>
22. </template>
```

3.19.14.5 La vue [FormCalculImpot]

La vue **[FormCalculImpot]** évolue comme suit :

```
1.  <!-- définition HTML de la vue -->
2.    <template>
3.    <!-- formulaire HTML -->
4.    <b-form @submit.prevent="calculerImpot" class="mb-3">
5.      <!-- message sur 12 colonnes sur fond bleu -->
6.      <b-row>
7.        <b-col sm="12">
8.          <b-alert show variant="primary">
9.            <h4>Remplissez le formulaire ci-dessous puis validez-le</h4>
10.         </b-alert>
11.       </b-col>
12.     </b-row>
13.     <!-- éléments du formulaire -->
14.     <!-- première ligne -->
15.     <b-form-group label="Etes-vous marié(e) ou pacsé(e) ?">
16.       <!-- boutons radio sur 5 colonnes-->
17.       <b-col sm="5">
18.         <b-form-radio v-model="marié" value="oui">Oui</b-form-radio>
19.         <b-form-radio v-model="marié" value="non">Non</b-form-radio>
20.       </b-col>
21.     </b-form-group>
22.     <!-- deuxième ligne -->
23.     <b-form-group label="Nombre d'enfants à charge" label-for="enfants">
24.       <b-form-input
25.         type="text"
26.         id="enfants"
27.         placeholder="Indiquez votre nombre d'enfants"
28.         v-model="enfants"
29.         :state="enfantsValide"
30.       ></b-form-input>
31.       <!-- message d'erreur éventuel -->
32.       <b-form-invalid-feedback :state="enfantsValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
33.     </b-form-group>
34.     <!-- troisème ligne -->
35.     <b-form-group
36.       label="Salaire annuel net imposable"
37.       label-for="salaire"
38.       description="Arrondissez à l'euro inférieur"
39.     >
40.       <b-form-input
41.         type="text"
42.         id="salaire"
43.         placeholder="Salaire annuel"
44.         v-model="salaire"
45.         :state="salaireValide"
46.       ></b-form-input>
47.       <!-- message d'erreur éventuel -->
48.       <b-form-invalid-feedback :state="salaireValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
49.     </b-form-group>
50.     <!-- quatrième ligne, bouton [submit] -->
51.     <b-col sm="3">
52.       <b-button type="submit" variant="primary" :disabled="formInvalide">Valider</b-button>
53.     </b-col>
54.   </b-form>
55. </template>
```

Commentaires

- lignes 15, 23, 35 : on a supprimé l'attribut **[label-cols]** ;

Par ailleurs, on fait évoluer les tests de validité :

```
1.  ...
2.  // état interne calculé
3.    computed: {
4.      // validation du formulaire
```

```
5.      formInvalide() {
6.        return (
7.          // salaire invalide
8.          !this.salaire.match(/^\s*\d+\s*$/) ||
9.          // ou enfants invalide
10.         !this.enfants.match(/^\s*\d+\s*$/) ||
11.         // ou données fiscales pas obtenues
12.         !this.$métier.taxAdminData
13.       );
14.     },
15.     // validation du salaire
16.     salaireValide() {
17.       // doit être numérique >=0
18.       return Boolean(
19.         this.salaire.match(/^\s*\d+\s*$/) || this.salaire.match(/^\s*$/)
20.       );
21.     },
22.     // validation des enfants
23.     enfantsValide() {
24.       // doit être numérique >=0
25.       return Boolean(
26.         this.enfants.match(/^\s*\d+\s*$/) || this.enfants.match(/^\s*$/)
27.       );
28.     }
29. },
30. ...
```

Commentaires

- ligne 19 : lorsque rien n'a été saisi, la saisie est considérée comme valide. Cela permet d'avoir une saisie valide lorsque la vue est initialement affichée. Dans la version précédente, la saisie apparaissait initialement comme erronée ;
- ligne 26 : idem ;
- lignes 5-14 : le bouton de validation n'est actif que si les deux saisies contiennent quelque chose et sont valides ;

3.19.14.6 La vue `[Menu]`

La vue **`[Menu]`** évolue comme suit :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <b-card class="mb-3">
4.      <!-- menu Bootstrap vertical -->
5.      <b-nav vertical>
6.        <!-- options du menu -->
7.        <b-nav-item
8.          v-for="(option,index) of options"
9.          :key="index"
10.         :to="option.path"
11.         exact
12.         exact-active-class="active"
13.       >{{option.text}}</b-nav-item>
14.     </b-nav>
15.   </b-card>
16. </template>
```

Commentaires

- ligne 3 : on ajoute la balise <b-card> pour entourer le menu d'une fine bordure. Cela permet de mieux localiser le menu sur le smartphone ;

3.19.14.7 La vue `[ListeSimulations]`

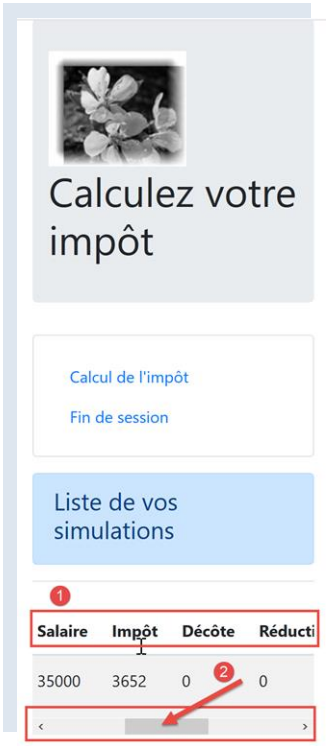
La vue **`[ListeSimulations]`** reste inchangée :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <!-- mise en page -->
5.      <Layout :left="true" :right="true">
6.        <!-- simulations dans colonne de droite -->
7.        <template slot="right">
8.          <template v-if="simulations.length==0">
9.            <!-- pas de simulations -->
10.           <b-alert show variant="primary">
```

```
11.             <h4>Votre liste de simulations est vide</h4>
12.           </b-alert>
13.         </template>
14.         <template v-if="simulations.length!=0">
15.           <!-- il y a des simulations -->
16.           <b-alert show variant="primary">
17.             <h4>Liste de vos simulations</h4>
18.           </b-alert>
19.           <!-- tableau des simulations -->
20.           <b-table striped hover responsive :items="simulations" :fields="fields">
21.             <template v-slot:cell(action)="data">
22.               <b-button variant="link" @click="supprimerSimulation(data.index)">Supprimer</b-button>
23.             </template>
24.           </b-table>
25.         </template>
26.       </template>
27.       <!-- menu de navigation dans colonne de gauche -->
28.       <Menu slot="left" :options="options" />
29.     </Layout>
30.   </div>
31. </template>
```

Commentaires

- ligne 20 : on notera l'attribut **[responsive]** qui fait que l'affichage de la table s'adapte à la taille de l'écran :

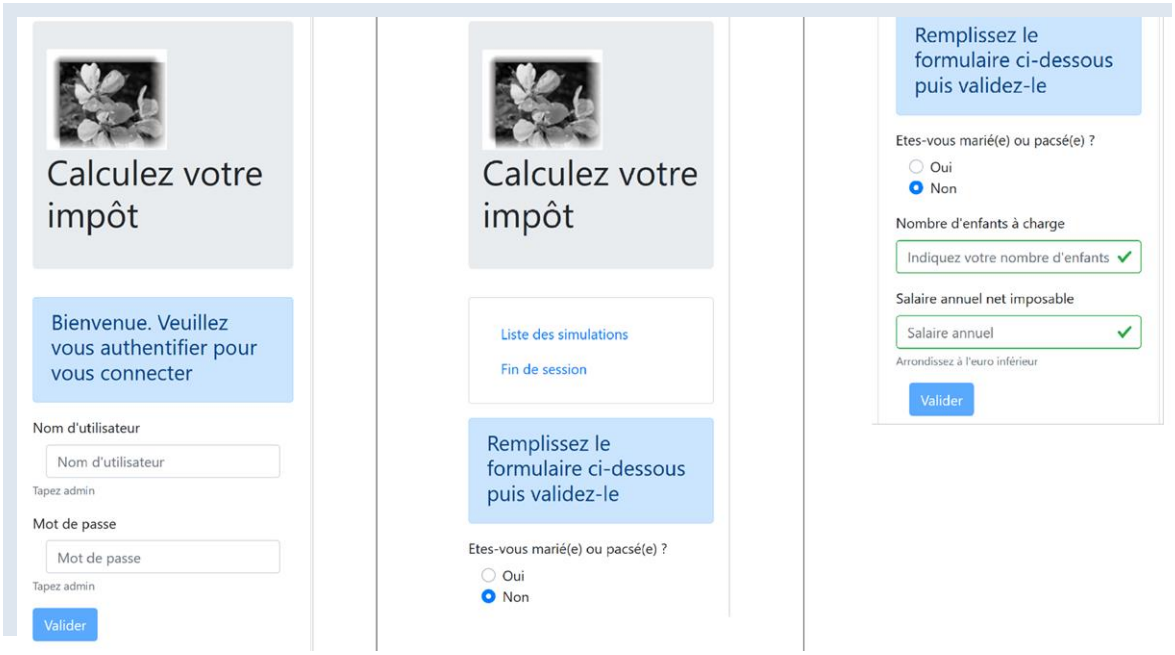


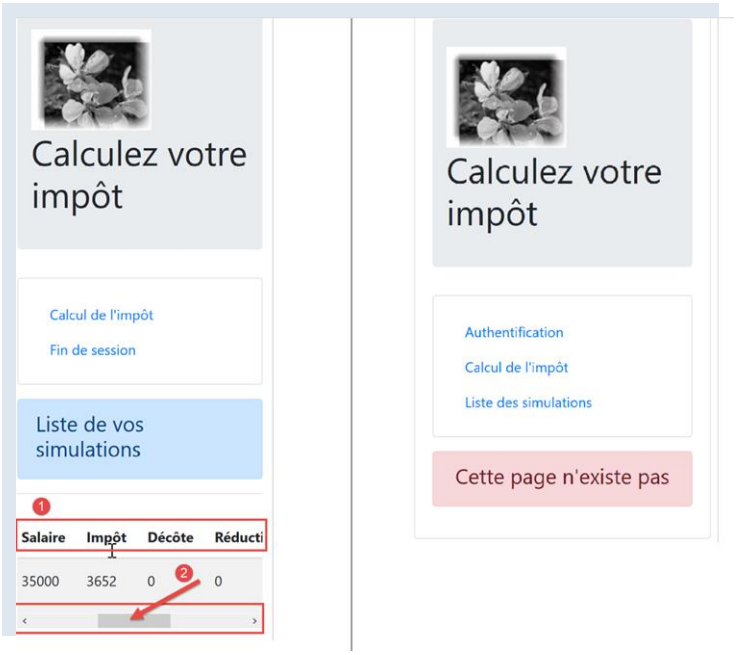
- en **[2]**, sur les petits écrans, une barre de défilement horizontal permet d'afficher la table ;

3.19.14.8 La vue [NotFound]

Elle reste inchangée.

3.19.14.9 Les vues sur mobile





Note : il y a sûrement possibilité d'obtenir des vues encore mieux adaptées au mobile. Je pense notamment au menu de navigation qui pourrait être amélioré mais il y a d'autres points. Ce document n'avait pas pour objectif premier la création d'une application mobile. Dans ce cas, on se serait peut-être tourné vers un framework comme Ionic |https://ionicframework.com|.

3.20 Déploiement de l'application client / serveur sur un service d'hébergement

Nous donnons ici les grandes lignes du déploiement de l'application client / serveur que nous avons développée sur un serveur OVH **[https://www.ovh.com/fr/]**. Le déploiement sur d'autres fournisseurs d'hébergement ne devrait pas être très différent. On veut simplement montrer que notre application se prête bien à ce déploiement.

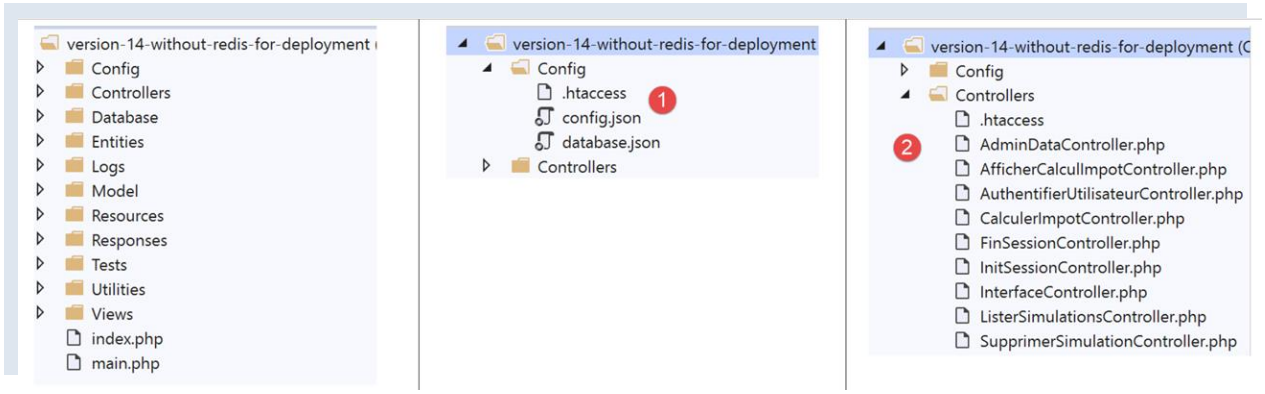
3.20.1 Déploiement du serveur

L'hébergement OVH visé est un hébergement basique :

1 un environnement PHP 7.3 ;

2 un SGBD MySQL ;
3 pas de serveur **[Redis]** ;

Le 3ième point nous oblige à modifier la version 14 de notre serveur de calcul de l'impôt.



Il nous faut modifier :

- les fichiers de configuration **[1]** ;
- les contrôleurs **[AdminDataController]** et **[CalculerImpotController]** pour tenir compte du fait qu'il n'y a pas de serveur **[Redis]** ;

Le fichier **[config.json]** évolue de la façon suivante :

```
1.  {
2.      "databaseFilename": "Config/database.json",
3.      "corsAllowed": false,
4.      "redisAvailable":false,
5.      "rootDirectory": "/.../www/apps/impot/serveur-php7",
6.      "relativeDependencies": [
7.
8.          "/Entities/BaseEntity.php",
9.          ...
10.         "/Controllers/AdminDataController.php"
11.     ],
12.     "absoluteDependencies": [
13.         "/.../vendor/autoload.php",
14.         "/.../vendor/predis/predis/autoload.php"
15.     ],
16.     "users": [
17.         {
18.             "login": "admin",
19.             "passwd": "admin"
20.         }
21.     ],
22.     ...
23. }
```

Commentaires

- ligne 4 : on introduit un booléen **[redisAvailable]** pour indiquer si on a accès ou non à un serveur **[Redis]** ;
- lignes 5, 13, 14 : les chemins absolus vont changer ;

Le fichier **[database.json]** évolue comme suit :

```
1.  {
2.      "dsn": "mysql:host=...;dbname=...",
3.      "id": "...",
4.      "pwd": "...",
5.      "tableTranches": "dbimpots_tbtranches",
6.      "colLimites": "limites",
7.      "colCoeffR": "coeffr",
8.      "colCoeffN": "coeffn",
9.      "tableConstantes": "dbimpots_tbconstantes",
10.     "colPlafondQfDemiPart": "plafondQfDemiPart",
11.     ...
12. }
```

Commentaires

- lignes 2-4 : l'identité de la base de données ainsi que les identifiants de son propriétaire vont changer ;

Le contrôleur **[AdminDataController]** évolue de la façon suivante :

```
1.  <?php
2.
3.  namespace Application;
4.
5.  // dépendances Symfony
6.  use \Symfony\Component\HttpFoundation\Response;
7.  use \Symfony\Component\HttpFoundation\Request;
8.  use \Symfony\Component\HttpFoundation\Session\Session;
9.  // alias de la couche [dao]
10. use \Application\ServerDaoWithSession as ServerDaoWithRedis;
11.
12. class AdminDataController implements InterfaceController {
13.
14.   // $config est la configuration de l'application
15.   // traitement d'une requête Request
16.   // utile la session Session et peut la modifier
17.   // $infos sont des informations supplémentaires propres à chaque contrôleur
18.   // rend un tableau [$statusCode, $état, $content, $headers]
19.   public function execute(
20.     array $config,
21.     Request $request,
22.     Session $session,
23.     array $infos = NULL): array {
24.
25.     // on doit avoir un unique paramètre GET
26.     $method = strtolower($request->getMethod());
27.     ...
28.
29.     // on peut travailler
30.     // Redis
31.     if ($config["redisAvailable"]) {
32.       \Predis\Autoloader::register();
33.       ...
34.     } else {
35.       try {
36.         // on va chercher les données fiscales en base de données
37.         $dao = new ServerDaoWithRedis($config["databaseFilename"], NULL);
38.         // taxAdminData
39.         $taxAdminData = $dao->getTaxAdminData();
40.       } catch (\Throwable $ex) {
41.         // ça s'est mal passé
42.         // retour résultat avec erreur au contrôleur principal
43.         $état = 1051;
44.         return [Response::HTTP_INTERNAL_SERVER_ERROR, $état,
45.           ["réponse" => utf8_encode($ex->getMessage())], []];
46.       }
47.     }
48.
49.     // retour résultat au contrôleur principal
50.     $état = 1000;
51.     return [Response::HTTP_OK, $état, ["réponse" => $taxAdminData], []];
52.   }
53.
54. }
```

Commentaires

- ligne 31 : on teste désormais si on a ou non un serveur **[Redis]** ;
- lignes 32-34 : si oui, le code précédent est repris dans son intégralité ;
- lignes 35-46 : sinon, les données de l'administration fiscale sont prises dans la base de données ;

Le contrôleur **[CalculerImpotController]** qui lui également a besoin des données de l'administration fiscale évolue de façon identique.

Ceci fait. Le déploiement sur le serveur OVH a consisté à faire du FTP. On a téléchargé sur OVH :

- la version **[vuejs-14-without-redis]** ;
- le dossier **[vendor]** qui contient toutes les dépendances du serveur **[vuejs-14-without-redis]** ;

Le transfert FTP fait, on a généré les tables nécessaires au serveur avec le script SQL suivant :

```
1.  -- phpMyAdmin SQL Dump
2.  -- version 4.8.5
3.  -- https://www.phpmyadmin.net/
4.  --
5.  -- Host: localhost:3306
6.  -- Generation Time: Oct 12, 2019 at 07:45 AM
7.  -- Server version: 5.7.24
8.  -- PHP Version: 7.2.11
9.
10. SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
11. SET AUTOCOMMIT = 0;
12. START TRANSACTION;
13. SET time_zone = "+00:00";
14.
15.
16. /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
17. /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
18. /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
19. /*!40101 SET NAMES utf8mb4 */;
20.
21. --
22. -- Table structure for table `dbimpots_tbconstantes`
23. --
24.
25. CREATE TABLE `dbimpots_tbconstantes` (
26.   `id` int(11) NOT NULL,
27.   `plafondQfDemiPart` decimal(10,2) NOT NULL,
28.   `plafondRevenusCelibatairePourReduction` decimal(10,2) NOT NULL,
29.   `plafondRevenusCouplePourReduction` decimal(10,2) NOT NULL,
30.   `valeurReducDemiPart` decimal(10,2) NOT NULL,
31.   `plafondDecoteCelibataire` decimal(10,2) NOT NULL,
32.   `plafondDecoteCouple` decimal(10,2) NOT NULL,
33.   `plafondImpotCelibatairePourDecote` decimal(10,2) NOT NULL,
34.   `plafondImpotCouplePourDecote` decimal(10,2) NOT NULL,
35.   `abattementDixPourcentMax` decimal(10,2) NOT NULL,
36.   `abattementDixPourcentMin` decimal(10,2) NOT NULL
37. ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
38.
39. --
40. -- Dumping data for table `dbimpots_tbconstantes`
41. --
42.
43. INSERT INTO `dbimpots_tbconstantes` (`id`, `plafondQfDemiPart`, `plafondRevenusCelibatairePourReduction`,
    `plafondRevenusCouplePourReduction`, `valeurReducDemiPart`, `plafondDecoteCelibataire`,
    `plafondDecoteCouple`, `plafondImpotCelibatairePourDecote`, `plafondImpotCouplePourDecote`,
    `abattementDixPourcentMax`, `abattementDixPourcentMin`) VALUES
44. (8, '1551.00', '21037.00', '42074.00', '3797.00', '1196.00', '1970.00', '1595.00', '2627.00', '12502.00',
    '437.00');
45.
46. -- --------------------------------------------------------
47.
48. --
49. -- Table structure for table `dbimpots_tbtranches`
50. --
51.
52. CREATE TABLE `dbimpots_tbtranches` (
53.   `id` int(11) NOT NULL,
54.   `limites` decimal(10,2) NOT NULL,
55.   `coeffR` decimal(10,2) NOT NULL,
56.   `coeffN` decimal(10,2) NOT NULL
57. ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
58.
59. --
60. -- Dumping data for table `dbimpots_tbtranches`
61. --
62.
63. INSERT INTO `dbimpots_tbtranches` (`id`, `limites`, `coeffR`, `coeffN`) VALUES
64. (36, '9964.00', '0.00', '0.00'),
65. (37, '27519.00', '0.14', '1394.96'),
66. (38, '73779.00', '0.30', '5798.00'),
67. (39, '156244.00', '0.41', '13913.69'),
```

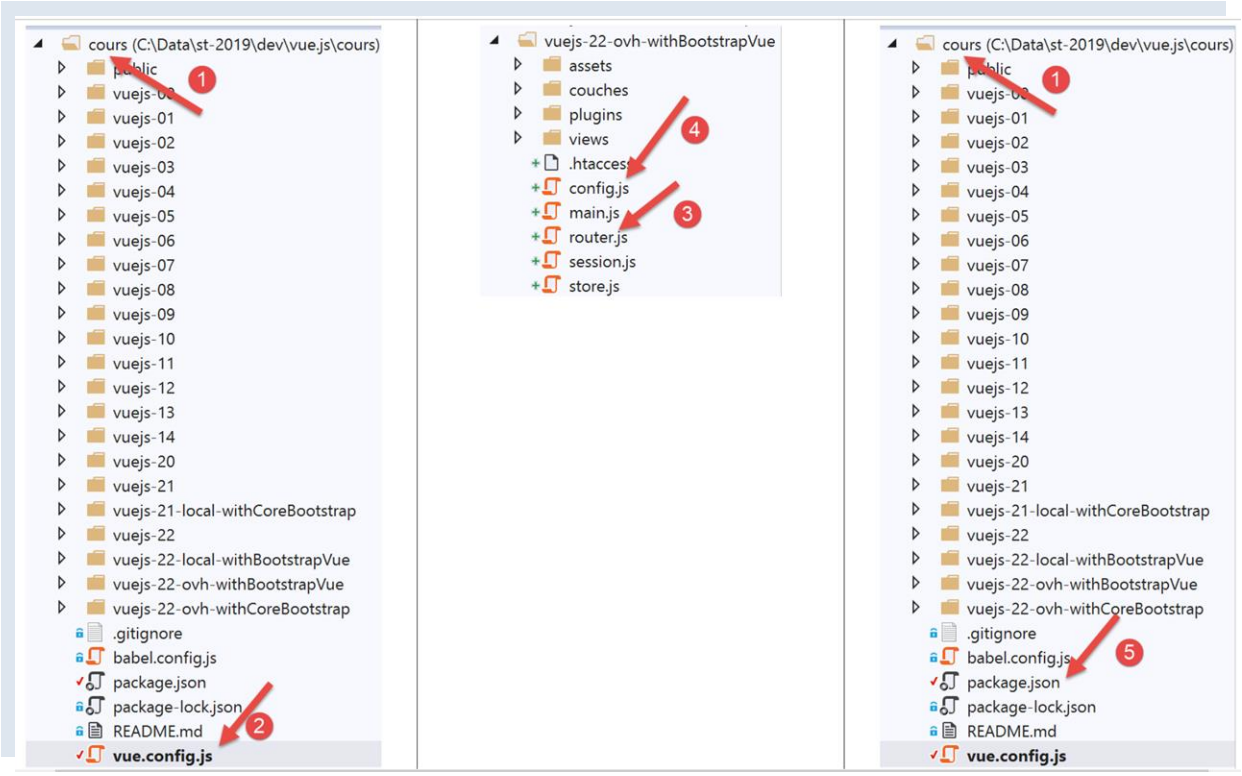
```
(40, '0.00', '0.45', '20163.45');

--
-- Indexes for dumped tables
--

--
-- Indexes for table `dbimpots_tbconstantes`
--
ALTER TABLE `dbimpots_tbconstantes`
  ADD PRIMARY KEY (`id`);

--
-- Indexes for table `dbimpots_tbtranches`
--
ALTER TABLE `dbimpots_tbtranches`
  ADD PRIMARY KEY (`id`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `dbimpots_tbconstantes`
--
ALTER TABLE `dbimpots_tbconstantes`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;

--
-- AUTO_INCREMENT for table `dbimpots_tbtranches`
--
ALTER TABLE `dbimpots_tbtranches`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=41;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Lorsque tout ceci a été fait, on a adapté les fichiers **[config.json, database.json]** à leur nouvel environnement.

3.20.2 Déploiement du client [Vue.js]

Il a été décidé de déployer le client **[Vue.js]** à l'URL **[http://machine/apps/impot/client-vuejs/]**. Cela a entraîné les modifications suivantes :

A la racine du **[workspace]** de **[VSCode]** on a créé le fichier **[vue.config.js]** suivant :



Le fichier **[vue.config.js]** est le suivant :

```
1.  // vue.config.js
2.  module.exports = {
3.    // l'URL de service du client [vuejs] du serveur de calcul de l'impôt
4.    publicPath: '/apps/impot/client-vuejs/'
5.  }
```

Le fichier **[router.js] [3]** a été également modifié :

```
1.  // imports
2.  import Vue from 'vue'
3.  import VueRouter from 'vue-router'
4.  ...
5.
6.  // plugin de routage
7.  Vue.use(VueRouter)
8.
9.  // les routes de l'application
10. const routes = [
11.   ...
12. ]
13.
14. // le routeur
15. const router = new VueRouter({
16.   // les routes
17.   routes,
18.   // le mode d'affichage des URL
19.   mode: 'history',
20.   // l'URL de base de l'application
21.   base: '/apps/impot/client-vuejs/'
22. })
23.
24. // vérification des routes
25. router.beforeEach((to, from, next) => {
26.   ...
27. })
28.
29. // export du router
30. export default router
```

Commentaires

- ligne 21 : la base des URL a été modifiée ;

Le fichier **[config.js]** est modifié de la façon suivante :

```
1.  // utilisation de la bibliothèque [axios]
2.  const axios = require('axios');
3.  // timeout des requêtes HTTP
4.  axios.defaults.timeout = 5000;
5.  // la base des URL du serveur de calcul de l'impôt
6.  // le schéma [https] pose des problèmes à Firefox parce que le serveur de calcul
7.  // de l'impôt envoie un certificat autosigné. ok avec Chrome et Edge. Safari pas testé.
8.  // avec Firefox c'est possible en demandant l'URL ci-dessous directement et en disant à Firefox
9.  // que vous acceptez le risque d'un certificat non signé. Ensuite le client [vuejs] fonctionnera.
10. axios.defaults.baseURL = 'http://.../apps/impot/serveur-php7';
11. // on va utiliser des cookies
12. axios.defaults.withCredentials = true;
13.
14. // export de la configuration
15. export default {
16.   // objet [axios]
17.   axios: axios,
18.   // délai maximal d'inactivité de la session : 5 mn = 300 s = 300000 ms
19.   duréeSession: 300000
20. }
```

Commentaires

- ligne 10 : on met l'URL du serveur de calcul de l'impôt ;

La version de production du projet a été générée avec la commande **[build]** du fichier **[package.json] [5]** suivant :

```
1.  {
2.    "name": "vuejs",
3.    "version": "0.1.0",
4.    "private": true,
5.    "scripts": {
6.      "serve": "vue-cli-service serve vuejs-22/main.js",
7.      "build": "vue-cli-service build vuejs-22-ovh-withBootstrapVue/main.js",
8.      "lint": "vue-cli-service lint"
9.    },
10.   ...
11. }
```

Ceci fait, le dossier **[dist]** qui contenait la version de production générée a été 'uploadée' sur le serveur OVH dans le dossier **[/.../apps/impot]** puis renommé **[client-vuejs]** pour que le code du client soit dans le dossier **[/.../apps/impot/client-vuejs/]** comme il était prévu. Puis dans ce dossier nous avons téléchargé le fichier **[.htaccess]** suivant :

```
1.  <IfModule mod_rewrite.c>
2.    RewriteEngine On
3.    RewriteBase /apps/impot/client-vuejs/
4.    RewriteRule ^index\.html$ - [L]
5.    RewriteCond %{REQUEST_FILENAME} !-f
6.    RewriteCond %{REQUEST_FILENAME} !-d
7.    RewriteRule . /apps/impot/client-vuejs/index.html [L]
8.  </IfModule>
```

ceci parce que le serveur web d'OVH utilisé ici est un serveur Apache. Pour d'autres types de serveurs, on se reportera à la documentation |https://cli.vuejs.org/guide/deployment.html|.

L'application serveur PHP 7 peut être testée |**ici**|.
Le client **[Vue.js]** peut être testé |**ici**|.

3.20.3 Conclusion

La version **[vuejs-21]** n'était pas indispensable. On avait vu que la version **[vuejs-20]** résistait correctement aux URL tapées par l'utilisateur. Néanmoins la nouvelle version amène un confort supplémentaire à l'utilisateur. Il peut naviguer en tapant des URL. L'application lui propose alors la vue qui convient le mieux à l'état actuel (la session) de l'application. Par ailleurs, la version **[vuejs-22]** amène des améliorations pour l'affichage de l'application sur mobiles.

INTRODUCTION

AU FRAMEWORK NUXT.JS

PAR L'EXEMPLE

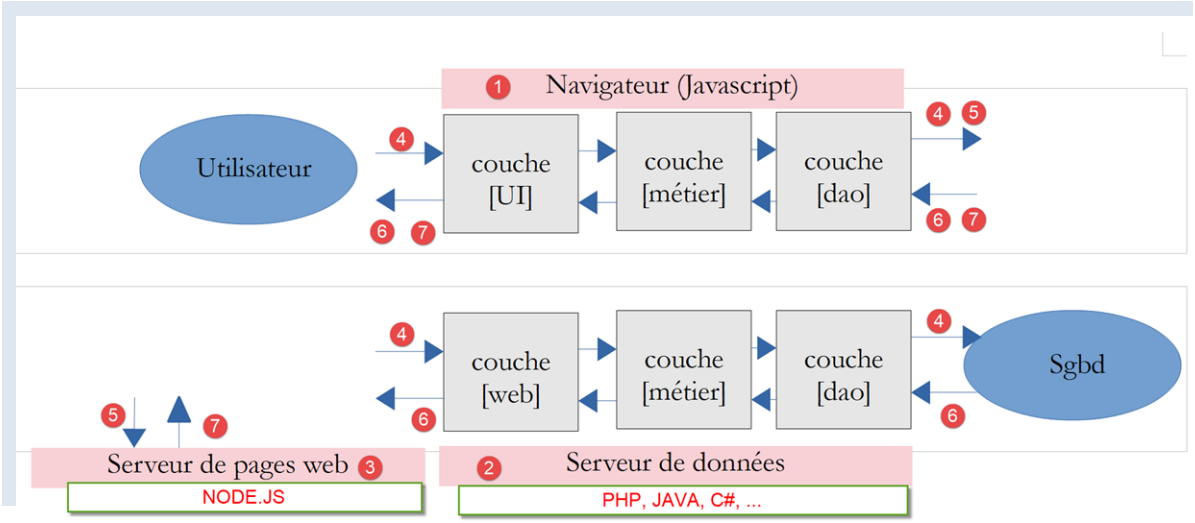
Serge Tahé, octobre 2019

4 Introduction au framework NUXT.JS par l'exemple

4.1 Présentation du cours

Cette section document poursuit le travail fait sur le |framework VUE.JS|.

Elle s'intéresse à l'architecture suivante :



- en **[1]**, un navigateur web affiche des pages web **[5, 7]** isues d'un serveur **[3]** à destination d'un utilisateur. Ces pages contiennent du Javascript implémentant un client d'un service web de données **[2]** ainsi qu'un client d'un serveur de fragments de pages web **[3]** ;
- en **[2]**, le serveur web est un serveur de données. Il peut être écrit dans n'importe quel langage. Il ne produit pas de pages web au sens classique (HTML, CSS, Javascript) sauf peut-être la 1ère fois. Mais cette 1ère page peut être obtenue d'un serveur web classique **[3]** (pas un serveur de données). Le Javascript de la page initiale va alors générer les différentes pages web de l'application en obtenant les données **[4]** à afficher, auprès du serveur web qui agit comme un serveur de données **[2]**. Il peut également obtenir des fragments de page web **[5]** pour habiller ces données auprès du serveur de pages web **[3]** ;
- en **[4]**, l'utilisateur initie une action ;
- en **[6,7]** : il reçoit des données habillées par un fragment de page web ;

Le framework **[nuxt.js]** |https://fr.nuxtjs.org/| va nous permettre d'implémenter le fonctionnement suivant :

- la 1ère page de l'application est délivrée par le serveur **[node.js]** **[3]**. Par ailleurs les autres pages de l'application sont également présentes sur ce même serveur. Elles sont délivrées lorsque l'utilisateur tape leur URL à la main dans le navigateur. Ces pages embarquent une application **[vue.js]** (approximativement) ;
- une fois la 1ère page chargée dans le navigateur, l'application se comporte comme une application **[vue.js]** classique. Dans notre schéma ci-dessus, elle va alors dialoguer avec le serveur de données **[2]** ;

Au final, l'application se comporte comme une application **[vue.js]** sauf pour la 1ère page et lorsque l'utilisateur tape des URL à la main. Dans ces cas, la page est cherchée sur le serveur **[3]**. Lorsque qu'un moteur de recherche demande les différentes pages de l'aplication, il reçoit les pages du serveur **[3]**. Celles-ci ont pu être optimisées pour le SEO (Search Engine Optimization). Dans une application **[vue.js]**, le moteur de recherche reçoit une page avec peu de signification SEO. Par exemple, dans l'application du client du serveur de calcul de l'impôt du document **[3]**, la page reçue par le navigateur lors du démarrage de l'application était la suivante :

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.    <head>
4.      <meta charset="utf-8">
5.      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.      <meta name="viewport" content="width=device-width,initial-scale=1.0">
7.      <link rel="icon" href="/client-vuejs-impot/favicon.ico">
8.      <title>vuejs</title>
9.    <link href="/client-vuejs-impot/app.js" rel="preload" as="script"></head>
10.   <body>
11.     <noscript>
12.       <strong>We're sorry but vuejs doesn't work properly without JavaScript enabled. Please enable it to
    continue.</strong>
13.     </noscript>
```

```
14.     <div id="app"></div>
15.     <!-- built files will be auto injected -->
16.   <script type="text/javascript" src="/client-vuejs-impot/app.js"></script></body>
17. </html>
```

C'est l'unique page chargée par le navigateur. Toutes les autres pages de l'application sont générées dynamiquement par le Javascript sans l'aide du navigateur. Certains moteurs de recherche se contentent de cette page. D'autres vont plus loin en exécutant le Javascript contenu dans la page (ligne 9 ci-dessus). Une autre page est alors obtenue. Celle-ci peut contenir une opération asynchrone pour aller chercher les données que la page va afficher. Dans ce cas les moteurs de recherche n'attendent pas. On se retrouve alors avec une page incomplète. On se rappelle peut-être que c'est le cas de notre client **[vue.js]** du serveur de calcul de l'impôt : de façon asynchrone, il initialise au cours du chargement de la 1ère page, une session jSON avec le serveur de calcul de l'impôt. Dans ce cas précis, cela n'influe pas sur la page récupérée par le moteur de recherche. Pour d'autres applications, cela pourrait être pénalisant en termes SEO.

Avec **[nuxt.js]** on peut servir au moteur de recherche une page plus signifiante pour chacune des pages de l'application.

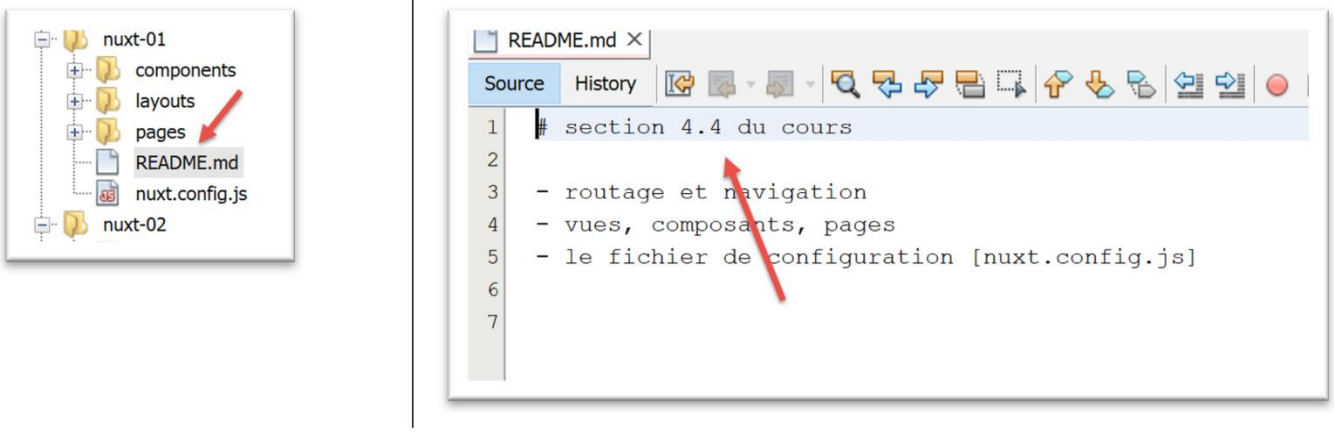
Les scripts de ce document sont commentés et leur exécution console reproduite. Des explications supplémentaires sont parfois fournies. Le document nécessite une lecture active : pour comprendre un script, il faut à la fois lire son code, ses commentaires et ses résultats d'exécution.

Les exemples du document sont disponibles |ici|.
L'application serveur PHP 7 peut être testée |ici|.

Serge Tahé, décembre 2019

Le contenu du document **[NUXT.JS]** est le suivant :

Certains lecteurs préfèreront lire, modifier et tester le code plutôt que de lire un cours. Dans chaque dossier des codes de ce document on a mis un fichier **[README.md]** résumant le contenu du dossier et faisant le lien avec le cours :



4.2 L'environnement de travail

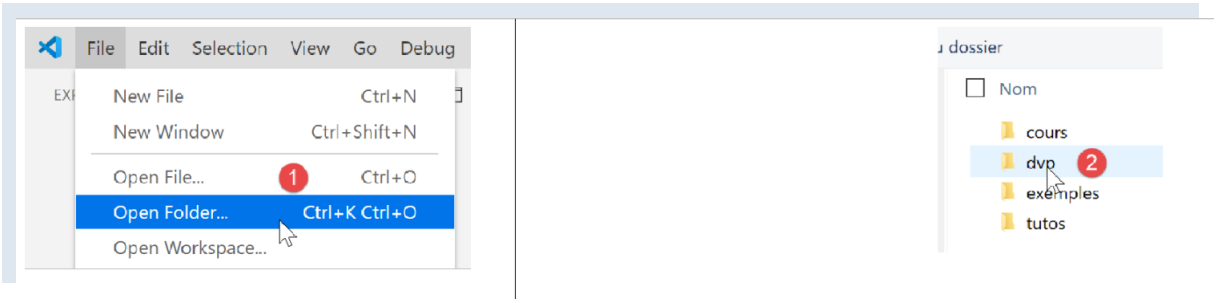
Nous utiliserons le même environnement de travail que celui présenté dans les documents :

- Introduction au langage **ECMASCRIPT 6** par l'exemple ;
- Introduction au framework **VUE.JS** par l'exemple ;

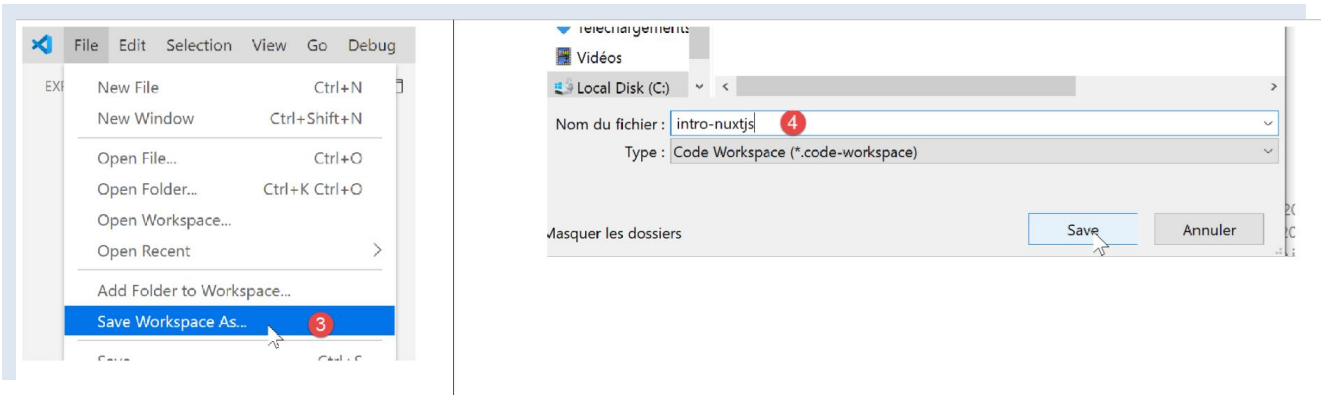
4.3 Une première application [nuxt.js]

4.3.1 Création de l'application

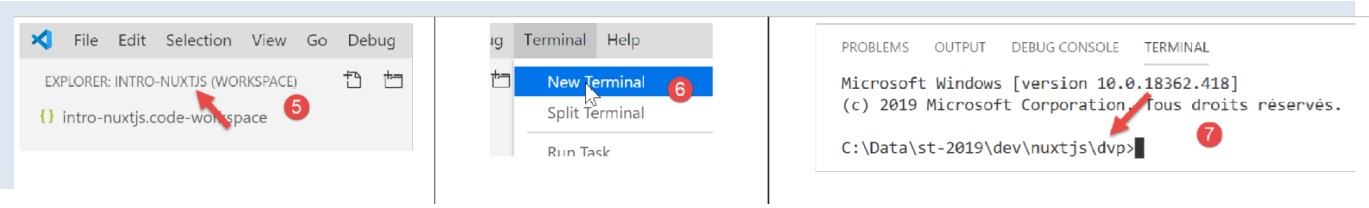
Pour nos développements **[nuxt.js]** nous continuons à utiliser VS Code. Nous avons créé un dossier **[dvp]** vide dans lequel nous allons mettre nos exemples. Puis nous ouvrons ce dossier :



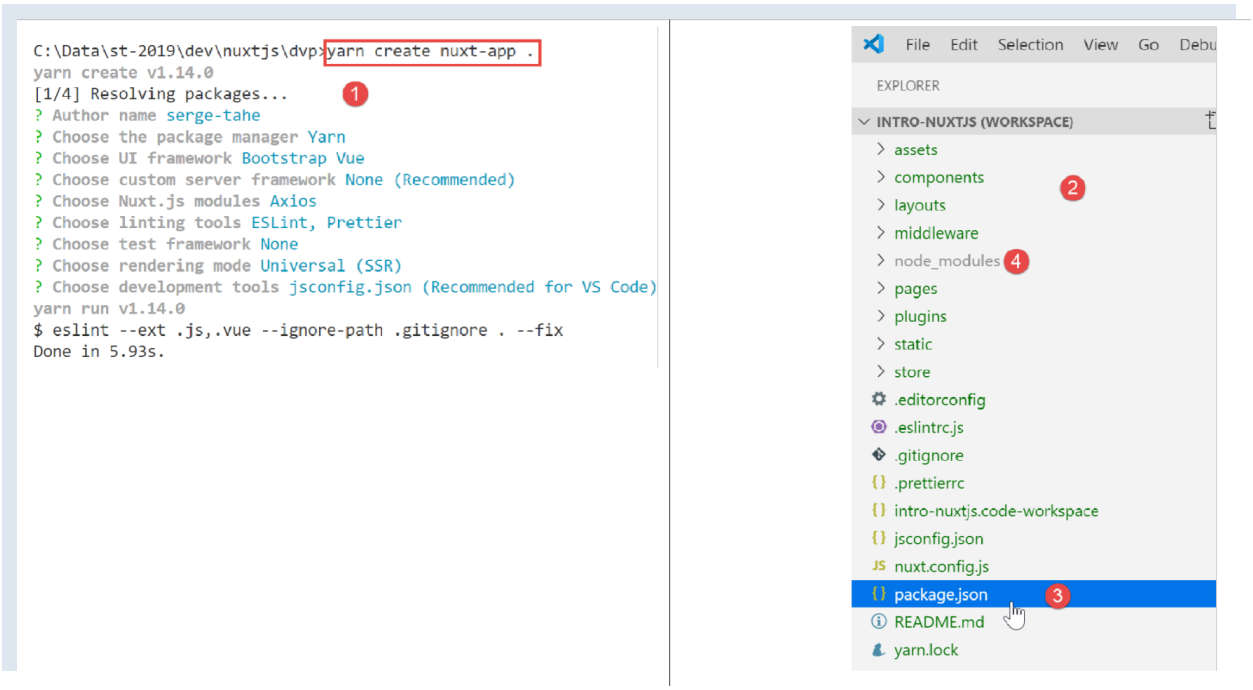
Nous sauvegardons l'espace de travail sous le nom **[intro-nuxtjs] [3-5]** :



Nous ouvrons un terminal **[6-7]** :



Jusqu'à maintenant, nous avons utilisé le gestionnaire de packages Javascript **[npm]**. Pour changer, nous allons ici utiliser le gestionnaire **[yarn]**. Celui-ci est installé, comme **[npm]**, avec les récentes versions de **[node.js]**. Pour créer une première application **[nuxt]**, nous utilisons la commande **[yarn create nuxt-app <dossier>] [1]**. La commande va demander un certain nombre d'informations sur le projet à générer puis, celles-ci obtenues, va le générer **[2]** :



En **[2]**, toute une arborescence de fichiers a été créée. Le fichier **[package.json]** donne la liste des biblothèques Javascript téléchargées dans le dossier **[node-modules] [4]** :

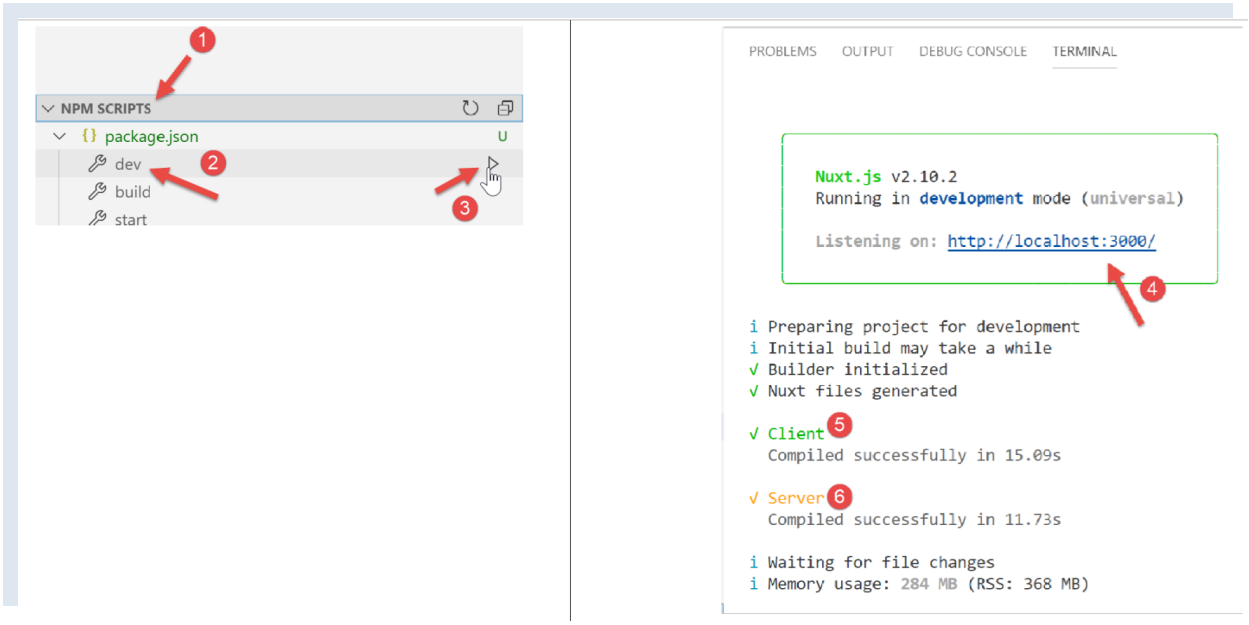
```
1.  {
2.    "name": "nuxt-intro",
3.    "version": "1.0.0",
4.    "description": "nuxt-intro",
5.    "author": "serge-tahe",
6.    "private": true,
7.    "scripts": {
8.      "dev": "nuxt",
9.      "build": "nuxt build",
10.     "start": "nuxt start",
11.     "generate": "nuxt generate",
12.     "lint": "eslint --ext .js,.vue --ignore-path .gitignore ."
13.   },
14.   "dependencies": {
15.     "nuxt": "^2.0.0",
16.     "bootstrap-vue": "^2.0.0",
17.     "bootstrap": "^4.1.3",
18.     "@nuxtjs/axios": "^5.3.6"
19.   },
20.   "devDependencies": {
21.     "@nuxtjs/eslint-config": "^1.0.1",
22.     "@nuxtjs/eslint-module": "^1.0.0",
23.     "babel-eslint": "^10.0.1",
24.     "eslint": "^6.1.0",
25.     "eslint-plugin-nuxt": ">=0.4.2",
26.     "eslint-config-prettier": "^4.1.0",
27.     "eslint-plugin-prettier": "^3.0.1",
28.     "prettier": "^1.16.4"
29.   }
```

```
30. }
```

Ce fichier reflète les réponses données à la commande **[create nuxt-app]** pour définir le projet créé (novembre 2019). Le lecteur peut avoir un fichier **[package.json]** différent :

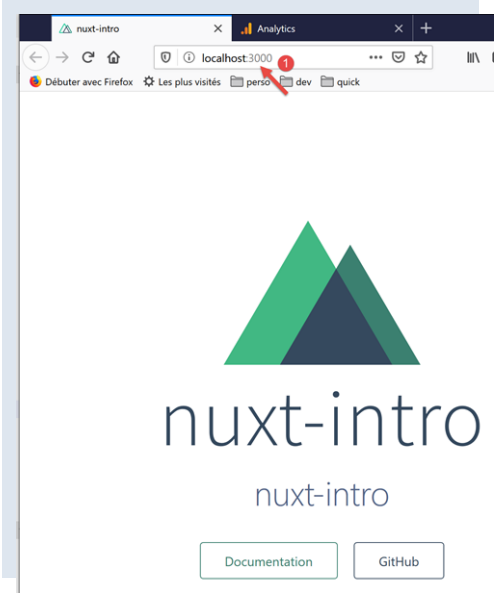
- il a pu donner des réponses différentes aux questions ;
- la commande **[create nuxt-app]** aura évolué depuis l'écriture de ce document : les dépendances et les versions auront changé ;

La ligne 8 du script est la commande qui lance l'application :



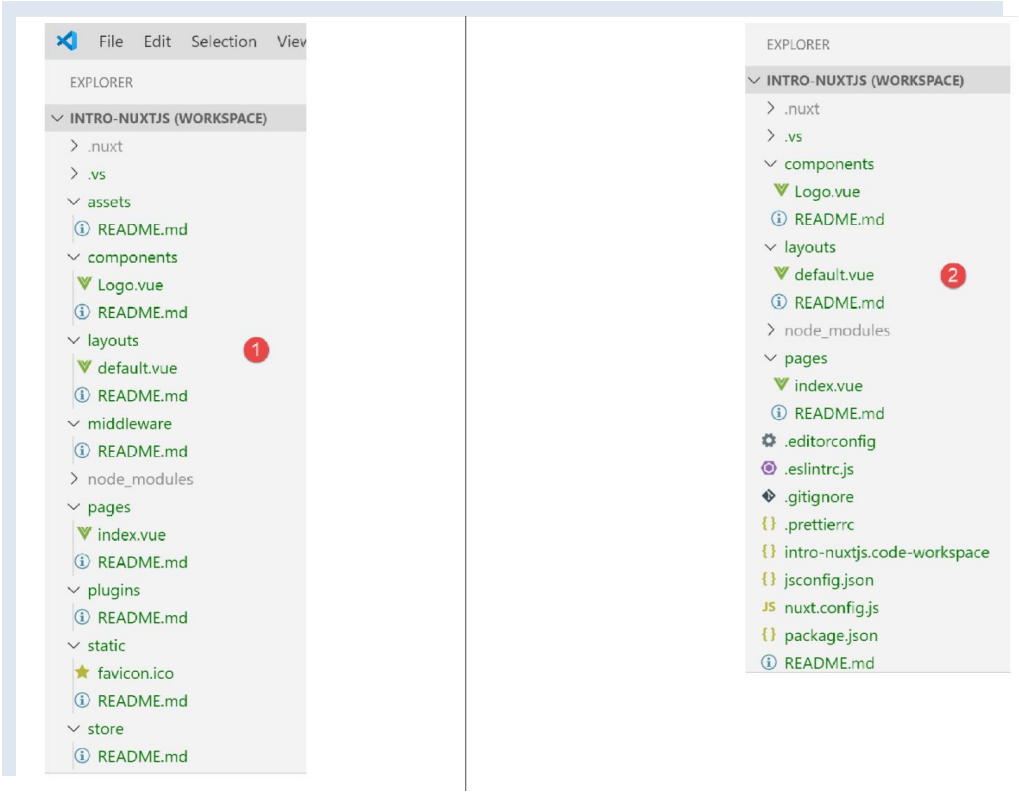
- en **[4]**, on voit que l'application est disponible à l'URL **[localhost:3000]** ;
- en **[5-6]**, on voit que l'application donne naissance à un serveur **[6]** et à un client (de ce serveur) **[5]** ;

Demandons l'URL **[http://localhost:3000/]** dans un navigateur :



4.3.2 Description de l'arborescence d'une application [nuxt]

Reprenons l'arborescence de l'application créée :



Le rôle des dossiers est le suivant :

assets	ressources non compilées de l'application (images, ...) ;
static	les fichiers de ce dossier seront disponibles à la racine de l'application. On met dans ce dossier des fichiers qu'on doit trouver à la racine de l'application comme par exemple le fichier **[robots.txt]** destiné aux moteurs de recherche ;
components	les composants **[vue]** de l'application utilisés dans les **[layouts]** et les **[pages]** ;
layouts	les composants **[vue]** de l'application servant de mise en page des **[pages]** ;
pages	les composants **[vue]** affichés par les différentes routes de l'application. On pourrait les appeler les vues de l'application. Les pages jouent un rôle particulier dans **[nuxt]** : les routes sont créées dynamiquement à partir de l'arborescence trouvée dans le dossier **[pages]** ;
middleware	les scripts exécutés à chaque changement de route. Ils permettent de contrôler celles-ci ;
plugins	porte un nom prêtant à confusion. Peut contenir des plugins mais également des scripts classiques. Les scripts trouvés dans ce dossier sont exécutés au démarrage de l'application ;
store	s'il contient un script **[index.js]** alors celui-ci définit une instance du store de **[Vuex]** ;

Si un dossier est vide, on peut le supprimer de l'arborescence. Ci-dessus, les dossiers **[assets, static, middleware, plugins, store]** peuvent être supprimés **[2]**.

4.3.3 Le fichier de configuration `[nuxt.config]`

L'exécution de l'application est contrôlée par le fichier **[nuxt.config.js]** suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: process.env.npm_package_name || '',
```

```
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: process.env.npm_package_description || ''
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: { color: '#fff' },
23.   /*
24.    ** Global CSS
25.    */
26.   css: [],
27.   /*
28.    ** Plugins to load before mounting the App
29.    */
30.   plugins: [],
31.   /*
32.    ** Nuxt.js dev-modules
33.    */
34.   buildModules: [
35.     // Doc: https://github.com/nuxt-community/eslint-module
36.     '@nuxtjs/eslint-module'
37.   ],
38.   /*
39.    ** Nuxt.js modules
40.    */
41.   modules: [
42.     // Doc: https://bootstrap-vue.js.org
43.     'bootstrap-vue/nuxt',
44.     // Doc: https://axios.nuxtjs.org/usage
45.     '@nuxtjs/axios'
46.   ],
47.   /*
48.    ** Axios module configuration
49.    ** See https://axios.nuxtjs.org/options
50.    */
51.   axios: {},
52.   /*
53.    ** Build configuration
54.    */
55.   build: {
56.     /*
57.      ** You can extend webpack config here
58.      */
59.     extend(config, ctx) {}
60.   }
61. }
```

- ligne 2 : le type d'application générée :
  - **[universal]** : application client / serveur. Au chargement initial de l'application ainsi qu'à chaque rafraîchissement de page dans le navigateur, le serveur est sollicité pour délivrer la page ;
  - **[sap]** : application de type **[Single Page Application]** : un serveur délivre initialement la totalité de l'application. Ensuite le client opère seul, même en cas de rafraîchissement d'une page dans le navigateur ;
- lignes 6-18 : définissent l'entête HTML <head> des différentes pages de l'application :
- ligne 7 : la balise <title> du titre des pages ;
- lignes 8-16 : les balises <meta> ;
- ligne 17 : les balises <link>

Dans l'application générée, la balise <head> est la suivante (code source de la page affichée dans le navigateur) :

```
1.  <title>nuxt-intro</title>
2.  <meta data-n-head="ssr" charset="utf-8">
3.  <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
4.  <meta data-n-head="ssr" data-hid="description" name="description" content="nuxt-intro">
5.  <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
6.  <link rel="preload" href="/_nuxt/runtime.js" as="script">
7.  <link rel="preload" href="/_nuxt/commons.app.js" as="script">
8.  <link rel="preload" href="/_nuxt/vendors.app.js" as="script">
```

```
9.  <link rel="preload" href="/_nuxt/app.js" as="script">
```

Maintenant, modifions le fichier **[nuxt.config]** de la façon suivante :

```
1.  head: {
2.      title: 'Introduction à [nuxt.js]',
3.      meta: [
4.        { charset: 'utf-8' },
5.        { name: 'viewport', content: 'width=device-width, initial-scale=1' },
6.        {
7.          hid: 'description',
8.          name: 'description',
9.          content: 'ssr routing loading asyncdata middleware plugins store'
10.       }
11.     ],
12.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
13. },
```

Lorsque nous réexécutons l'application, la balise <head> est devenue la suivante (code source de la page affichée dans le navigateur) :

```
1.     <head >
2.       <title>Introduction à [nuxt.js]</title>
3.       <meta data-n-head="ssr" charset="utf-8">
4.       <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
5.       <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading
     asyncdata middleware plugins store">
6.       <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
7.       <link rel="preload" href="/_nuxt/runtime.js" as="script">
8.       <link rel="preload" href="/_nuxt/commons.app.js" as="script">
9.       <link rel="preload" href="/_nuxt/vendors.app.js" as="script">
10. <link rel="preload" href="/_nuxt/app.js" as="script">
```

Revenons au fichier **[nuxt.config]** :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      ...
8.    },
9.    /*
10.    ** Customize the progress-bar color
11.    */
12.   loading: { color: '#fff' },
13.   /*
14.    ** Global CSS
15.    */
16.   css: [],
17.   /*
18.    ** Plugins to load before mounting the App
19.    */
20.   plugins: [],
21.   /*
22.    ** Nuxt.js dev-modules
23.    */
24.   buildModules: [
25.     // Doc: https://github.com/nuxt-community/eslint-module
26.     '@nuxtjs/eslint-module'
27.   ],
28.   /*
29.    ** Nuxt.js modules
30.    */
31.   modules: [
32.     // Doc: https://bootstrap-vue.js.org
33.     'bootstrap-vue/nuxt',
34.     // Doc: https://axios.nuxtjs.org/usage
35.     '@nuxtjs/axios'
36.   ],
37.   /*
38.    ** Axios module configuration
39.    ** See https://axios.nuxtjs.org/options
40.    */
41.   axios: {},
```

```
42.   /*
43.    ** Build configuration
44.    */
45.   build: {
46.     /*
47.      ** You can extend webpack config here
48.      */
49.     extend(config, ctx) {}
50.   }
51. }
```

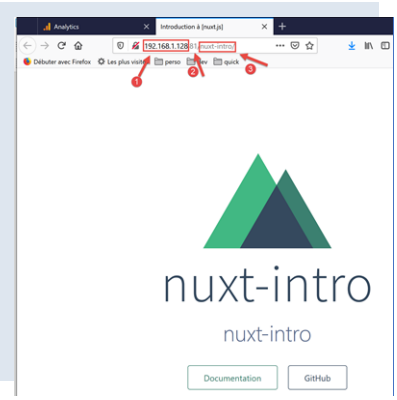
- ligne 12 : entre chaque route du client **[nuxt]**, une barre de chargement (loading) apparaît si le changement de route prend un peu de temps. La propriété **[loading]** permet de paramétrer cette barre de chargement, ici la couleur de la barre ;
- ligne 16 : les fichiers **[css]** globaux. Ils seront automatiquement inclus dans toutes les pages de l'application ;
- lignes 24-27 : les modules Javascript nécessaires à la compilation (build) de l'application ;
- lignes 31-36 : les modules Javascript utilisés par l'application ;
- ligne 41 : paramétrage de la bibliothèque **[axios]** lorsque celle-ci a été sélectionnée par l'utilisateur pour les dialogues HTTP avec des serveurs tiers ;
- lignes 45-50 : paramétrage de la compilation (build) du projet ;

On peut ajouter d'autres clés au fichier de configuration. On peut notamment paramétrer le port de service (3000 par défaut) et la racine du projet (par défaut, le dossier racine du projet). C'est ce que nlous faisons maintenant en ajoutant les clés suivantes :

```
1.  // répertoire du code source
2.    srcDir: '.',
3.    router: {
4.      // URL racine des pages de l'application
5.      base: '/nuxt-intro/'
6.    },
7.    // serveur
8.    server: {
9.      // port de service - par défaut 3000
10.     port: 81,
11.     // adresses réseau écoutées - par défaut localhost=127.0.0.1
12.     host: '0.0.0.0'
13. }
```

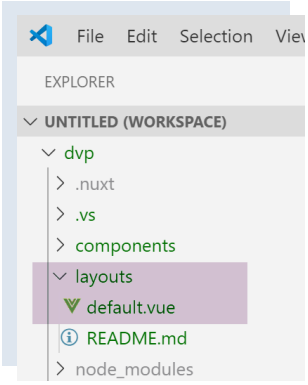
- ligne 2 : où trouver le code source du projet. On le trouve ici dans le dossier courant, ç-à-d au même niveau que le fichier **[nuxt.config.js]**. C'est la valeur par défaut ;
- lignes 8-13 : configurent le serveur (il ne faut pas oublier qu'une application **[nuxt]** de type **[universal]** est installée à la fois sur un serveur et un navigateur client de ce serveur) ;
- ligne 10 : les pages de l'application seront délivrées sur le port 81 du serveur ;
- ligne 12 : par défaut **[localhost]** (adresse réseau 127.0.0.1). Une machine peut avoir plusieurs adresses réseau si elle appartient à plusieurs réseaux. L'adresse 0.0.0.0 indique que le serveur web écoute toutes les adresses réseau de la machine ;
- lignes 3-6 : configurent le routeur de l'applicatio **[nuxt]** ;
- ligne 5 : les pages de l'application seront disponibles à l'URL **[http://localhost:81/nuxt-intro/]**;

Ajoutons ces lignes au fichier **[nuxt.config.js]** puis exécutons le projet (script npm dev). Le résultat est le suivant :



- en **[1]**, l'adresse de la machine sur un réseau public ;
- en **[2]**, le port de service ;
- en **[3]**, l'URL racine de l'application ;

4.3.4 Le dossier `[layouts]`



Le dossier **`[layouts]`** est destiné aux composants de mise en page. Par défaut, c'est le composant nommé **`[default.vue]`** qui est utilisé. Dans ce projet, celui-ci est le suivant :

```
<template>
  <div>
    <nuxt />
  </div>
</template>

<style>
html {
  font-family: 'Source Sans Pro', -apple-system, BlinkMacSystemFont, 'Segoe UI',
    Roboto, 'Helvetica Neue', Arial, sans-serif;
  font-size: 16px;
  word-spacing: 1px;
  -ms-text-size-adjust: 100%;
  -webkit-text-size-adjust: 100%;
  -moz-osx-font-smoothing: grayscale;
  -webkit-font-smoothing: antialiased;
  box-sizing: border-box;
}

*,
*:before,
*:after {
  box-sizing: border-box;
  margin: 0;
}

.button--green {
  display: inline-block;
  border-radius: 4px;
  border: 1px solid #3b8070;
  color: #3b8070;
  text-decoration: none;
  padding: 10px 30px;
}

.button--green:hover {
  color: #fff;
  background-color: #3b8070;
}

.button--grey {
  display: inline-block;
  border-radius: 4px;
  border: 1px solid #35495e;
  color: #35495e;
  text-decoration: none;
  padding: 10px 30px;
  margin-left: 15px;
}

.button--grey:hover {
  color: #fff;
  background-color: #35495e;
}
```

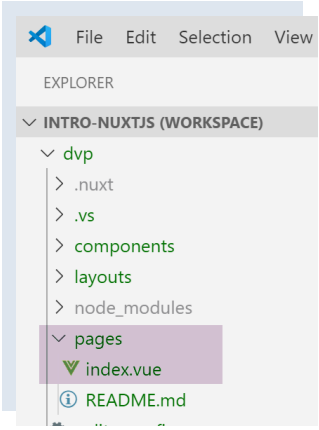
```
55. </style>
```

Commentaires

- lignes 1-5 : le **[template]** du composant ;
- ligne 3 : la balise <**nuxt** /> désigne la page courante du routage ;
- lignes 7-55 : le style embarqué par le composant de mise en page. Comme celui-ci contient la page courante du routage, ce style va s'appliquer à toutes les pages routées de l'application ;

On voit que le but premier de la page **[default.vue]** est ici d'appliquer un style aux pages routées.

4.3.5 Le dossier [pages]



Le dossier **[pages]** contient les vues routées, celles que voit l'utilisateur. La page **[index.vue]** est la page d'accueil de l'application. Avec **[nuxt.js]**, il n'y a pas de fichier de routage. Les routes sont déterminées à partir de la structure du dossier **[pages]**. Ici la présence d'un fichier **[index.vue]** va automatiquement créér une route appelée **[index]** et de chemin **[/index]** ramené à **[/]** puisqu'il s'agit de la page d'accueil. Ainsi la route suivante est créée :

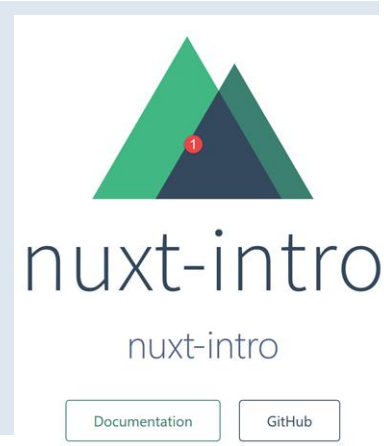
```
{ name : 'index', path : '/'}
```

Le fichier **[index.vue]** est ici le suivant :

```
1.  <template>
2.    <div class="container">
3.      <div>
4.        <logo />
5.        <h1 class="title">
6.          nuxt-intro
7.        </h1>
8.        <h2 class="subtitle">
9.          nuxt-intro
10.       </h2>
11.       <div class="links">
12.         <a href="https://nuxtjs.org/" target="_blank" class="button--green">
13.           Documentation
14.         </a>
15.         <a
16.           href="https://github.com/nuxt/nuxt.js"
17.           target="_blank"
18.           class="button--grey"
19.         >
20.           GitHub
21.         </a>
22.       </div>
23.     </div>
24.   </div>
25. </template>
26.
27. <script>
28. import Logo from '~/components/Logo.vue'
29.
30. export default {
31.   components: {
```

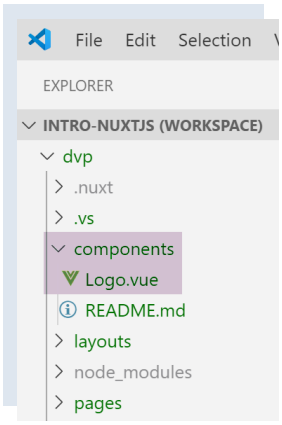
```
32.     Logo
33.   }
34. }
35. </script>
36.
37. <style>
38. .container {
39.   margin: 0 auto;
40.   min-height: 100vh;
41.   display: flex;
42.   justify-content: center;
43.   align-items: center;
44.   text-align: center;
45. }
46.
47. .title {
48.   font-family: 'Quicksand', 'Source Sans Pro', -apple-system, BlinkMacSystemFont,
49.     'Segoe UI', Roboto, 'Helvetica Neue', Arial, sans-serif;
50.   display: block;
51.   font-weight: 300;
52.   font-size: 100px;
53.   color: #35495e;
54.   letter-spacing: 1px;
55. }
56.
57. .subtitle {
58.   font-weight: 300;
59.   font-size: 42px;
60.   color: #526488;
61.   word-spacing: 5px;
62.   padding-bottom: 15px;
63. }
64.
65. .links {
66.   padding-top: 15px;
67. }
68. </style>
```

Le **[template]** des lignes 1-25 affichent la vue suivante :



L'image **[1]** est générée par la ligne 4 du **[template]**. On voit donc que la page utilise un composant appelé **[logo]**. Celui-ci est défini aux lignes 27-35 du script de la page. Ligne 28, la notation **[~]** désigne la racine du projet.

4.3.6 Le composant `[Logo]`



Le composant **`[Logo.vue]`** est le suivant :

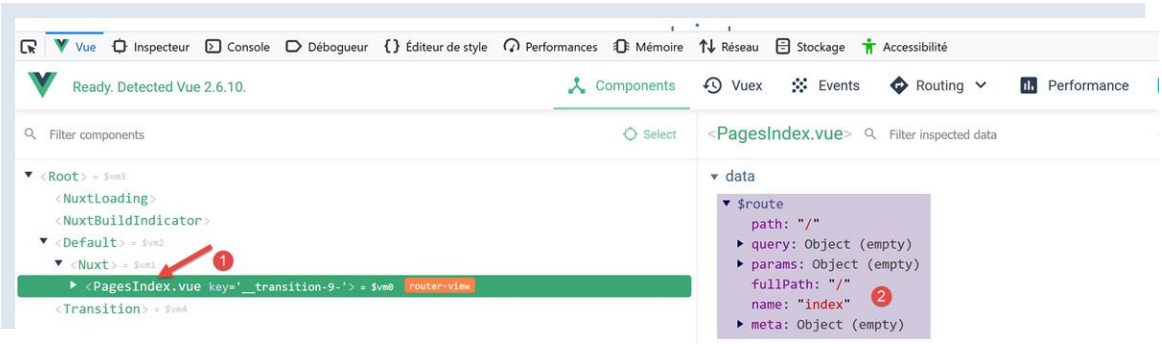
```
1.  <template>
2.    <div class="VueToNuxtLogo">
3.      <div class="Triangle Triangle--two" />
4.      <div class="Triangle Triangle--one" />
5.      <div class="Triangle Triangle--three" />
6.      <div class="Triangle Triangle--four" />
7.    </div>
8.  </template>
9.
10. <style>
11. .VueToNuxtLogo {
12.   display: inline-block;
13.   animation: turn 2s linear forwards 1s;
14.   transform: rotateX(180deg);
15.   position: relative;
16.   overflow: hidden;
17.   height: 180px;
18.   width: 245px;
19. }
20.
21. .Triangle {
22.   position: absolute;
23.   top: 0;
24.   left: 0;
25.   width: 0;
26.   height: 0;
27. }
28.
29. .Triangle--one {
30.   border-left: 105px solid transparent;
31.   border-right: 105px solid transparent;
32.   border-bottom: 180px solid #41b883;
33. }
34.
35. .Triangle--two {
36.   top: 30px;
37.   left: 35px;
38.   animation: goright 0.5s linear forwards 3.5s;
39.   border-left: 87.5px solid transparent;
40.   border-right: 87.5px solid transparent;
41.   border-bottom: 150px solid #3b8070;
42. }
43.
44. .Triangle--three {
45.   top: 60px;
46.   left: 35px;
47.   animation: goright 0.5s linear forwards 3.5s;
48.   border-left: 70px solid transparent;
49.   border-right: 70px solid transparent;
50.   border-bottom: 120px solid #35495e;
51. }
52.
```

```
53. .Triangle--four {
54.   top: 120px;
55.   left: 70px;
56.   animation: godown 0.5s linear forwards 3s;
57.   border-left: 35px solid transparent;
58.   border-right: 35px solid transparent;
59.   border-bottom: 60px solid #fff;
60. }
61.
62. @keyframes turn {
63.   100% {
64.     transform: rotateX(0deg);
65.   }
66. }
67.
68. @keyframes godown {
69.   100% {
70.     top: 180px;
71.   }
72. }
73.
74. @keyframes goright {
75.   100% {
76.     left: 70px;
77.   }
78. }
79. </style>
```

Ce composant est essentiellement constitué de styles et d'animations pour créer une image animée.

4.3.7 Vue DevTools

[Vue DevTools] est l'extension de navigateur qui permet d'inspecter les objets **[nuxt.js]** et **[vue.js]** dans le navigateur. Nous l'avons déjà utilisée dans le chapitre sur **[vue.js]**. Examinons ce que cet outil trouve lorsque la page d'accueil de notre application est affichée :



- en **[1]**, le composant **[PagesIndex]** désigne la page **[pages/index.vue]** ;
- on voit en **[2]** que ce composant a une propriété **[$route]** qui est la route qui a amené à la page **[index]** ;

Comme simple exercice, affichons cette route dans la console.

4.3.8 Modification de la page d'accueil

Nous allons modifier le fichier **[index.vue]**. Dans notre installation du projet, nous avons installé deux dépendances :

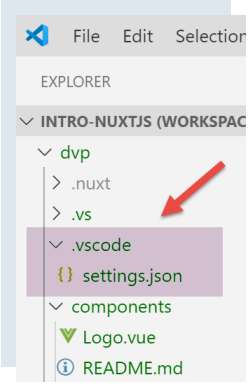
- **[eslint]** : qui vérifie la syntaxe des fichiers Javascript et des composants Vue. Si l'extension **[ESLint]** de VSCode a été installée, cette syntaxe est vérifiée lors de la frappe des textes et les erreurs sont immédiatement signalées ;
- **[prettier]** : qui formate les codes Javascript d'une façon standard ;

Ces dépendances sont inscrites dans le fichier **[package.json]** :

```
1.  "devDependencies": {
2.      "@nuxtjs/eslint-config": "^1.0.1",
3.      "@nuxtjs/eslint-module": "^1.0.0",
4.      "babel-eslint": "^10.0.1",
5.      "eslint": "^6.1.0",
6.      "eslint-config-prettier": "^4.1.0",
```

```
7.      "eslint-plugin-nuxt": ">=0.4.2",
8.      "eslint-plugin-prettier": "^3.0.1",
9.      "prettier": "^1.16.4"
10. }
```

J'ai pu remarquer (nov 2019) qu'avec l'installation faite par la commande **[yarn create nuxt-app]**, les outils **[eslint, prettier]** ne fonctionnent pas lors de la frappe des textes. Les erreurs ne sont signalées qu'à la compilation. Après quelques recherches, j'ai trouvé une configuration qui marche :



On installe à la racine du projet, un dossier **[.vscode]** avec dedans le fichier **[settings.json]** suivant :

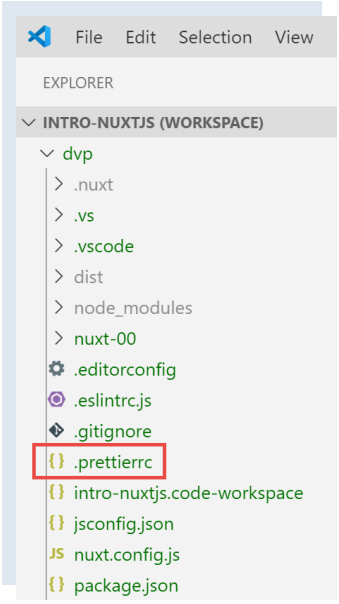
```
1.  {
2.    "eslint.validate": [
3.      {
4.        "language": "vue",
5.        "autoFix": true
6.      },
7.      {
8.        "language": "javascript",
9.        "autoFix": true
10.     }
11.   ],
12.   "eslint.autoFixOnSave": true,
13.   "editor.formatOnSave": false
14. }
```

- lignes 2-11 : indiquent que lorsque **[eslint]** valide les fichiers .vue et .js il doit corriger les erreurs qu'il peut corriger ;
- ligne 12 : lorsqu'un fichier est sauvegardé, **[eslint]** doit corriger les erreurs qu'il peut corriger ;
- ligne 13 : inhibe le formatage fait par défaut dans VSCode lors d'une sauvegarde. C'est **[prettier]** qui le fera ;

Avec cette configuration :

- les erreurs de syntaxe ou de formatage sont signalées dès la frappe des textes ;
- les erreurs de formatage sont automatiquement corrigées lors de la sauvegarde du fichier ;

La bibliothèque **[prettier]** est configurée par le fichier **[.prettierrc]** :



Ce fichier est par défaut le suivant :

```
1.  {
2.    "semi": false,
3.    "arrowParens": "always",
4.    "singleQuote": true
5.  }
```

- ligne 1 : pas de ; à la fin des instructions ;
- ligne 2 : si une fonction 'flèche' (arrow) a un unique paramètre, celui-ci est entouré de parenthèses ;
- ligne 3 : les chaînes de caractères sont entourées d'apostrophes (pas de guillemets) ;

Nous ajoutons les deux règles suivantes :

```
1.  {
2.    "semi": false,
3.    "arrowParens": "always",
4.    "singleQuote": true,
5.    "printWidth": 120,
6.    "endOfLine": "auto"
7.  }
```

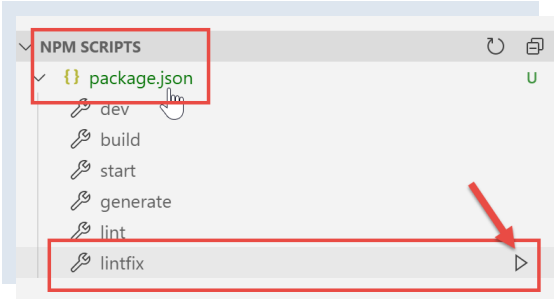
- ligne 5 : la ligne de code peut faire jusqu'à 120 caractères ;
- ligne 6 : la marque de fin de ligne peut être indifféremment CRLF (windows) ou LF (unix) ;

Enfin, le fichier **[package.json]** est modifié de la façon suivante :

```
1.  "scripts": {
2.      "dev": "nuxt",
3.      "build": "nuxt build",
4.      "start": "nuxt start",
5.      "generate": "nuxt generate",
6.      "lint": "eslint --ext .js,.vue --ignore-path .gitignore .",
7.      "lintfix": "eslint --fix --ext .js,.vue --ignore-path .gitignore ."
8.  },
```

- ligne 7 : nous ajoutons la commande **[lintfix]** qui est identique à la commande **[lint]** de la ligne 6 si ce n'est qu'elle a en plus le paramètre **[--fix]**. La commande **[lint]** vérifie la syntaxe et le format de tous les fichiers du projet et signale toute erreur. **[lintfix]** fera la même chose si ce n'est que les problèmes de formatage qui peuvent être corrigés le seront automatiquement. **[lintfix]** sera la commande à utiliser si la compilation échoue à cause de problèmes de formatage de fichiers ;

Ceci fait, nous modifions le fichier **[index.vue]** de la façon suivante :



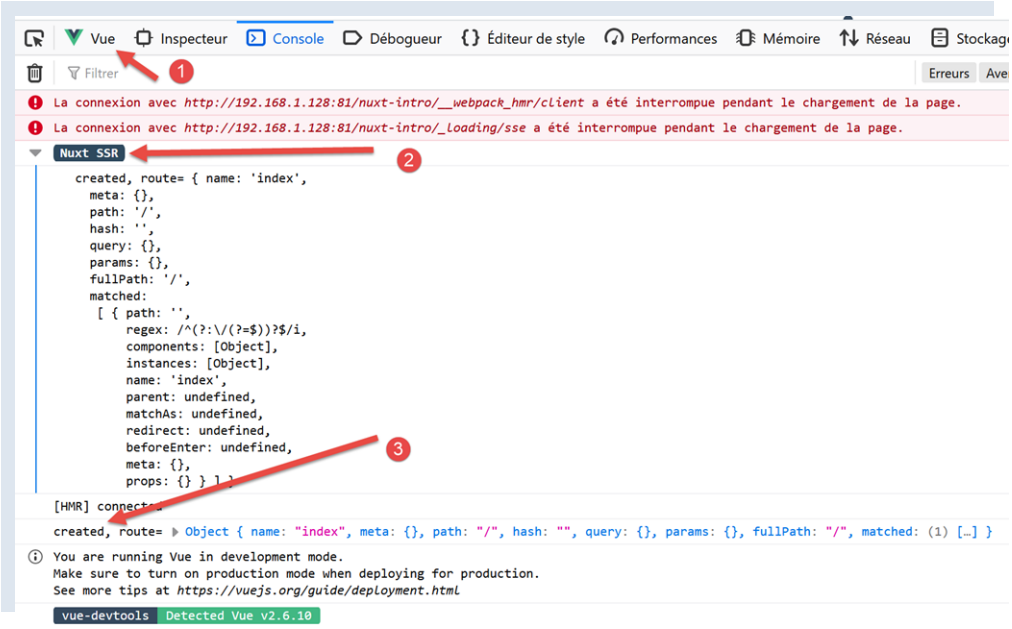

```
1.  <script>
2.  /* eslint-disable no-console */
3.  import Logo from '~/components/Logo.vue'
4.
5.  export default {
6.    components: {
7.      Logo
8.    },
9.    // cycle de vie
10.   created() {
11.     console.log('created, route=', this.$route)
12.   }
13. }
14. </script>
```

- lignes 10-12 : on ajoute la fonction **[created]** qui est automatiquement exécutée lorsque le composant a été créé ;
- ligne 11 : on affiche la route courante ;
- ligne 2 : un commentaire destiné à **[eslint]**. Sans ce commentaire, **[eslint]** signale une erreur ligne 11 : il ne veut pas d'instructions **[console]** dans les fonctions du cycle de vie. **[eslint]** est configurable. Nous allons garder sa configuration par défaut et nous utiliserons des commentaires tels que celui de la ligne 2 pour désactiver une règle précise de **[eslint]**. Nous utiliserons deux types de commentaires :

  /* désactivation règle **[eslint]** */ : désactivation d'une règle pour tout le fichier ;
  // désactivation règle **[eslint]** : désactivation d'une règle pour la ligne qui suit ;

Lors de la frappe, les erreurs sont signalées et une fonction **[Quick Fix]** disponible :

On exécute le projet :



- en **[1]**, l'onglet **[Vue]** des outils de développement du navigateur (F12) ;
- en **[2]** et **[3]**, l'affichage de la route ;

Pourquoi deux affichages et non un seul ?

Une application **[nuxt]** se décompose de deux éléments, un serveur et un client :

- le serveur fournit les pages de l'application au démarrage de celle-ci et puis à chaque fois qu'une page est rafraîchie dans le navigateur (F5) ou bien que l'utilisateur tape une URL de l'application à la main ;
- chaque page fournie par le navigateur contient la page demandée ainsi que le code Javascript de toute l'application qui est ensuite exécutée sur le navigateur. C'est le client. Tant qu'il n'y a pas rafraîchissement de page sur le navigateur, l'application fonctionne comme une application Vue classique en mode **[sap]** (Single Page Application). Dès que l'utilisateur provoque manuellement un rafraîchissement de page, celle-ci est demandée au serveur et on retourne à la phase 1 précédente.

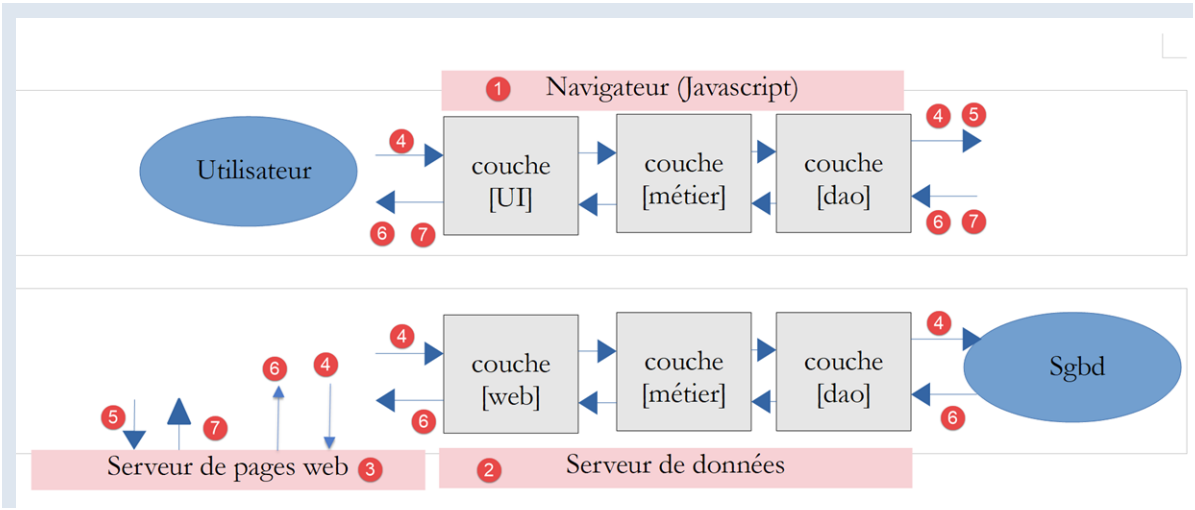
Ce qu'il faut comprendre, c'est que ce sont les **mêmes pages** du dossier **[pages]** qui sont fournies par le serveur ou le client. Pour cette raison, les concepteurs de **[nuxt]** appellent ce type de pages, des pages **isomorphiques**. Les mêmes pages **[.vue]** peuvent être interprétées à la fois par le client et le serveur. Prenons l'exemple de la page **[index]** :

```
1.  <template>
2.    <div class="container">
3.      <div>
4.        <logo />
5.        <h1 class="title">
6.          nuxt-intro
7.        </h1>
8.        <h2 class="subtitle">
9.          nuxt-intro
10.       </h2>
11.       <div class="links">
12.         <a href="https://nuxtjs.org/" target="_blank" class="button--green">
13.           Documentation
14.         </a>
15.         <a
16.           href="https://github.com/nuxt/nuxt.js"
17.           target="_blank"
18.           class="button--grey"
19.         >
20.           GitHub
21.         </a>
22.       </div>
23.     </div>
24.   </div>
25. </template>
26.
27. <script>
28. /* eslint-disable no-console */
29. import Logo from '~/components/Logo.vue'
```

```
30.
31. export default {
32.   components: {
33.     Logo
34.   },
35.   // cycle de vie
36.   created() {
37.     console.log('created, route=', this.$route)
38.   }
39. }
40. </script>
```

Comme c'est la page d'accueil, au démarrage de l'application elle est servie par le serveur. La page sur le serveur a également un cycle de vie, le même que celle d'une page **[Vue]** classique sauf pour les fonctions **[beforeMount, monted]** qui n'existent pas côté serveur. La fonction **[created]** est elle exécutée ce qui explique le 1er log. Cela signifie au passage que le serveur est capable d'exécuter des scripts Javascript. Ici et en général, ce serveur est un serveur **[node.js]**. Une fois la page créée sur le serveur, elle arrive sur le navigateur où elle subit de nouveau le cycle de vie. La fonction **[created]** est exécutée une seconde fois, ce qui donne le 2ième log.

L'architecture d'une application **[nuxt]** pourrait être la suivante :



- **[1]** : le navigateur qui héberge l'application **[nuxt]** lorsque celle-ci a été chargée sur le navigateur. C'est ce qu'on a appelé le client **[nuxt]** ;
- **[3]** : le serveur qui héberge initialement l'application **[nuxt]**. Celle-ci est chargée sur le navigateur **[1]** au démarrage de l'application et à chaque fois que l'utilisateur rafraîchit la page courante du navigateur ou tape à la main une URL de l'application. C'est là que se situe la différence de fonctionnement avec une application Vue classique. Avec celle-ci, une fois chargée sur le navigateur, le serveur n'était plus jamais sollicité par la suite. Une autre différence importante qu'on n'a pas pu voir pour l'instant est que le serveur d'une application Vue est un serveur statique, incapable d'interpréter les pages **[.vue]**, alors que celui d'une application Nuxt de type **[universal]** est un serveur Javascript. Avant d'envoyer une page au navigateur, le serveur peut exécuter des scripts et aller par exemple chercher des données sur le serveur **[2]** ;
- **[2]** : est le serveur qui fournit des données soit au client **[nuxt] [1]**, soit au serveur **[nuxt] [3]** ;

On peut dans le schéma ci-dessus distinguer trois sous-systèmes client / serveur :

- **[1, 3]** : héberge l'application **[nuxt]**. **[3]** la fournit au démarrage de l'application avec la page d'accueil et à chaque fois que l'utilisateur demande une page manuellement. **[1]** héberge l'application **[nuxt]** reçue de **[3]** qui fonctionne alos en mode **[SAP]** tant que les pages ne sont pas demandées manuellement à **[3]** ;
- **[1, 2]** : en mode **[SAP]**, le client **[nuxt]** récupère des données externes auprès d'un ou plusieurs serveurs ;
- **[3, 2]** : lors de la génération de la page demandée par l'utilisateur, le serveur **[3]** peut lui aussi récupérer des données externes auprès d'un ou plusieurs serveurs ;

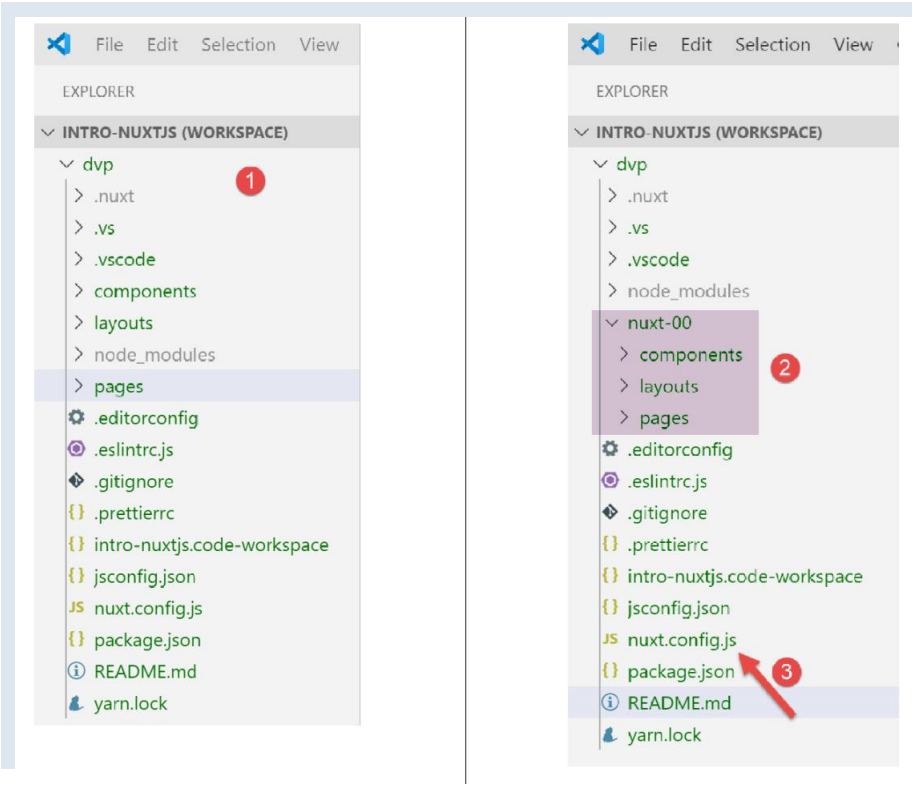
C'est donc le serveur **[3]** qui distingue une application **[nuxt]** d'une application **[vue]**. Ce serveur est sollicité à chaque fois que l'utilisateur demande une page manuellement. Il traite les mêmes pages **[.vue]** que le client **[vue] [1]**. C'est un serveur Javascript capable d'exécuter les scripts présents dans la page. Cela peut modifier par exemple la façon de générer la page d'accueil avec des données externes : là où une application **[vue]** obtient celles-ci forcément à partir du client **[1]**, ici elles peuvent être obtenues par le serveur **[3]** avant que la page ne soit envoyée au client. La page d'accueil devient ainsi signifiante et peut contribuer à améliorer le SEO de l'application.

Note : en mode développement les trois entités **[1, 2, 3]** sont souvent sur la même machine. Ce sera le cas ici pour tous nos exemples.

4.3.9 Déplacement du code source de l'application dans un dossier séparé

Par la suite, nous allons créer diverses applications **[nuxt]** dans le même dossier **[dvp]**. En effet, le dossier des dépendances **[node_modules]** généré pour chaque projet **[nuxt]** peut faire plusieurs centaines de méga-octets. On va créer divers dossiers **[nuxt-00, nuxt-01, ...]** dans le dossier **[dvp]** pour contenir le code source des exemples à tester. Puis nous utiliserons le fichier de configuration **[nuxt-config.js]** pour indiquer où se trouve le code source du projet **[dvp]** qui restera l'unique projet **[nuxt]** de ce tutoriel.

Nous déplaçons le code source de l'application générée initialement par la commande **[yarn create nuxt-app]** dans un dossier **[nuxt-00]** :



- en **[2]**, on a déplacé les dossiers **[components, layouts, pages]** dans un dossier **[nuxt-00]** ;
- en **[3]**, il nous faut modifier le fichier **[nuxt.config.js]** ;

Nous modifions le le fichier **[nuxt.config.js]** de la façon suivante :

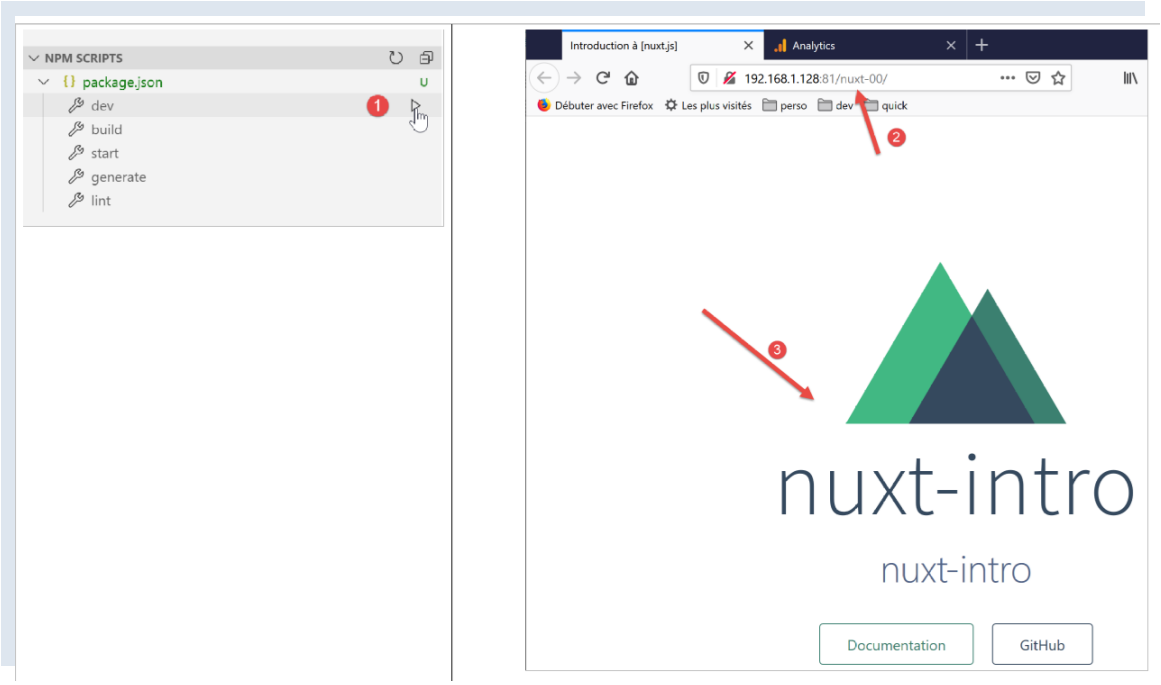
```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    ...
7.    /*
8.     ** Build configuration
9.     */
10.   build: {
11.     /*
12.      ** You can extend webpack config here
13.      */
14.     extend(config, ctx) {}
15.   },
16.   // répertoire du code source
17.   srcDir: 'nuxt-00',
18.   // routeur
19.   router: {
20.     // racine des URL de l'application
21.     base: '/nuxt-00/'
```

```
22.   },
23.   // serveur
24.   server: {
25.     // port de service, 3000 par défaut
26.     port: 81,
27.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
28.     // 0.0.0.0 = toutes les adresses réseau de la machine
29.     host: '0.0.0.0'
30.   }
31. }
```

Le fichier est modifié en deux points :

- ligne 17 : on indique que le code source du projet **[dvp]** est à trouver dans le dossier **[nuxt-00]** ;
- ligne 21 : on indique que l'URL racine de l'application est désormais **[/nuxt-00/]**. Ce changement n'était pas obligatoire. On pourrait ne pas mettre cette propriété et la racine des URL serait alors **[/]**. Ici, cela nous permettra de nous souvenir que le code source exécuté est celui du dossier **[nuxt-00]** ;

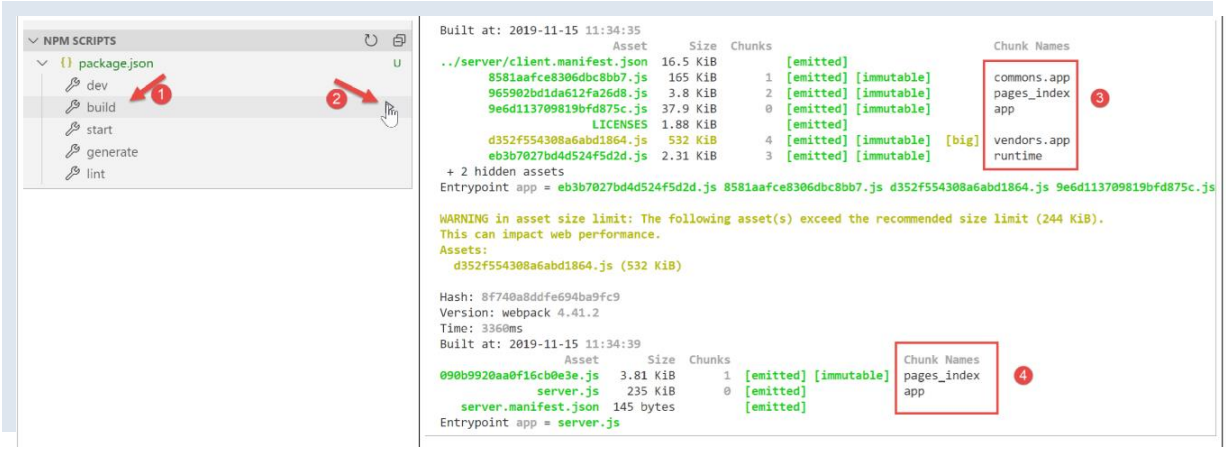
Ceci fait, le projet **[dvp]** est exécuté comme précédemment :



4.3.10 Déploiement de l'application [nuxt-00]

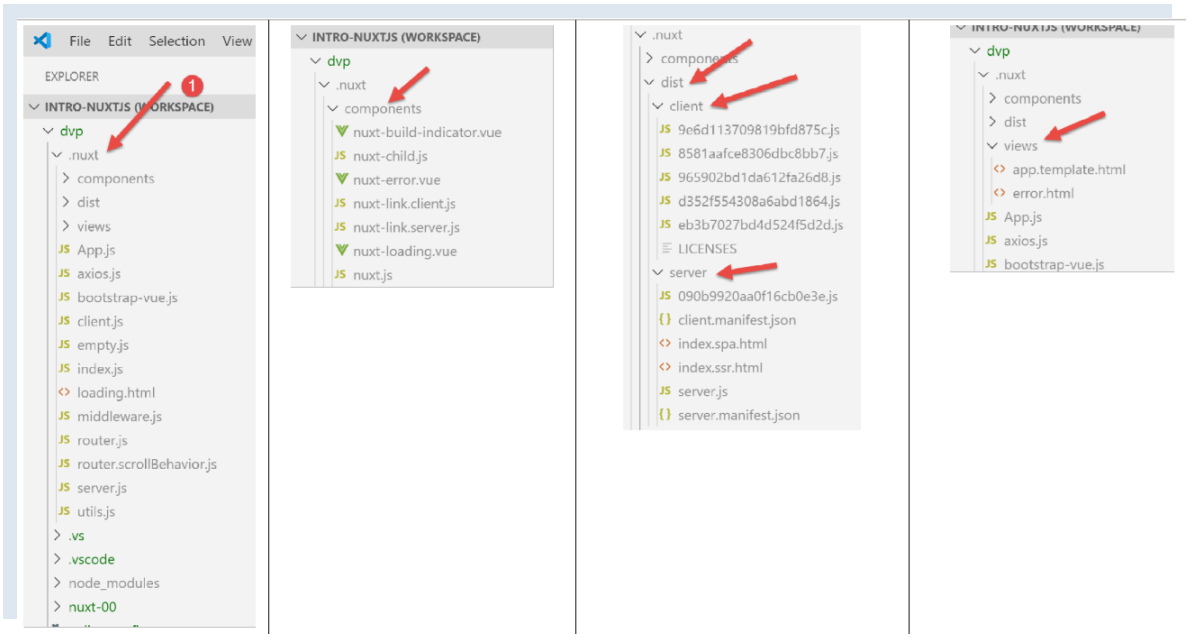
Nous allons exécuter l'application **[nuxt-00]** dans un environnement autre que l'environnement intégré de VSCode.

Tout d'abord nous compilons l'application :

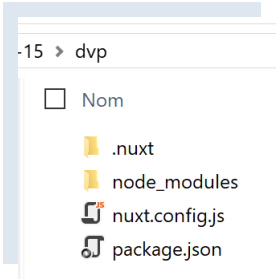


- en **[3]**, le résultat de la compilation du client. Sera exécuté par le navigateur ;
- en **[4]**, le résultat de la compilation du serveur. Sera exécuté par le serveur **[node.js]** ;

Le résultat de la compilation est placé dans le dossier **[.nuxt]** :



Nous copions les dossiers **[.nuxt, node_modules]** et les fichiers **[package.json, nuxt.config.js]** dans un dossier séparé :



Le fichier **[package.json]** est simplifié de la façon suivante :

```
1.  {
2.    "scripts": {
3.      "start": "nuxt start"
4.    }
5.  }
```

- on ne garde que le script **[start]** qui permet d'exécuter la version compilée du projet ;

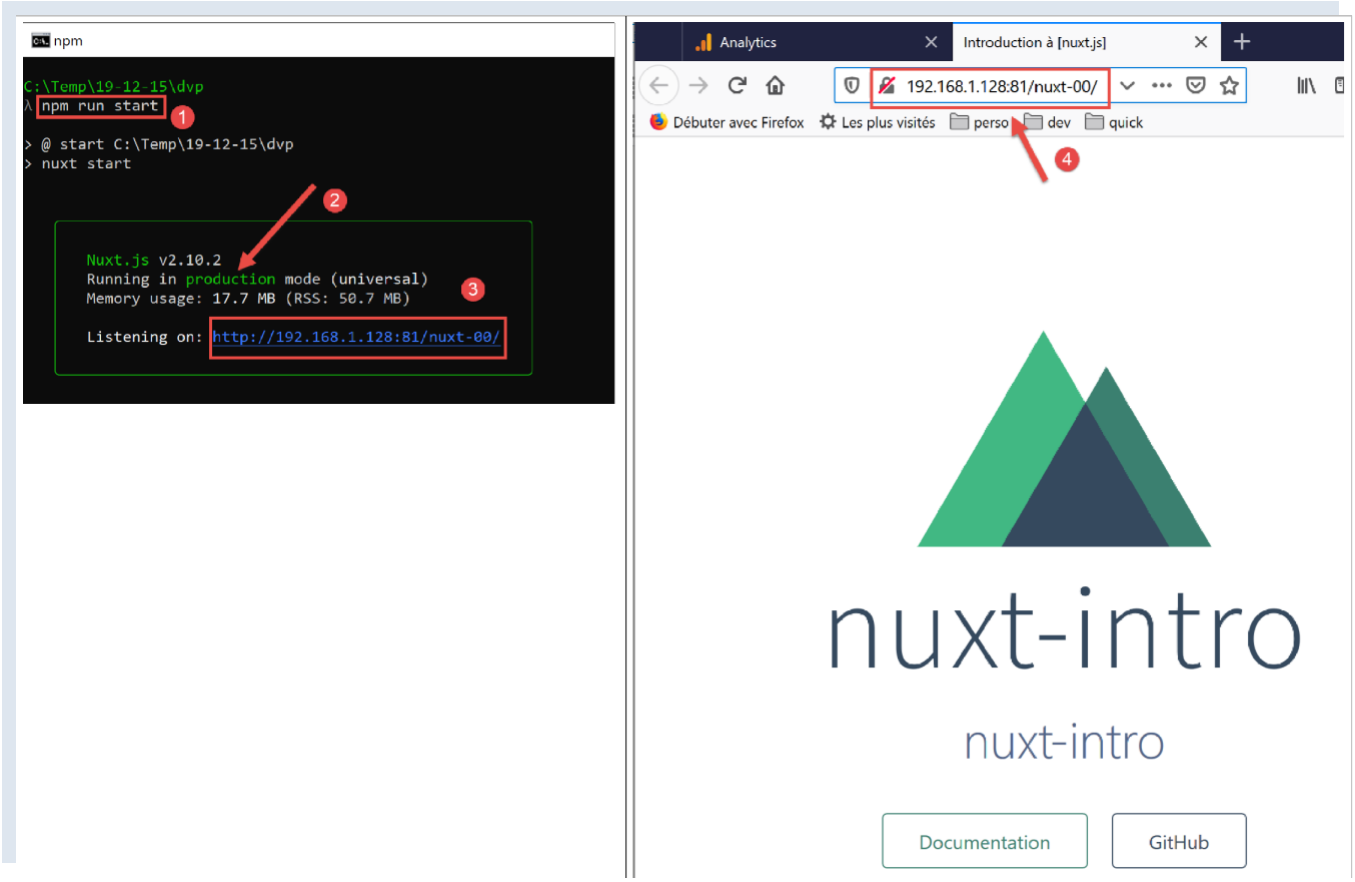
Le fichier **[nuxt.config.js]** est simplifié de la façon suivante :

```
1.  export default {
2.    // routeur
3.    router: {
4.      // racine des URL de l'application
5.      base: '/nuxt-00/'
6.    },
7.    // serveur
8.    server: {
9.      // port de service, 3000 par défaut
10.     port: 81,
11.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
12.     // 0.0.0.0 = toutes les adresses réseau de la machine
13.     host: '0.0.0.0'
14.   }
15. }
```

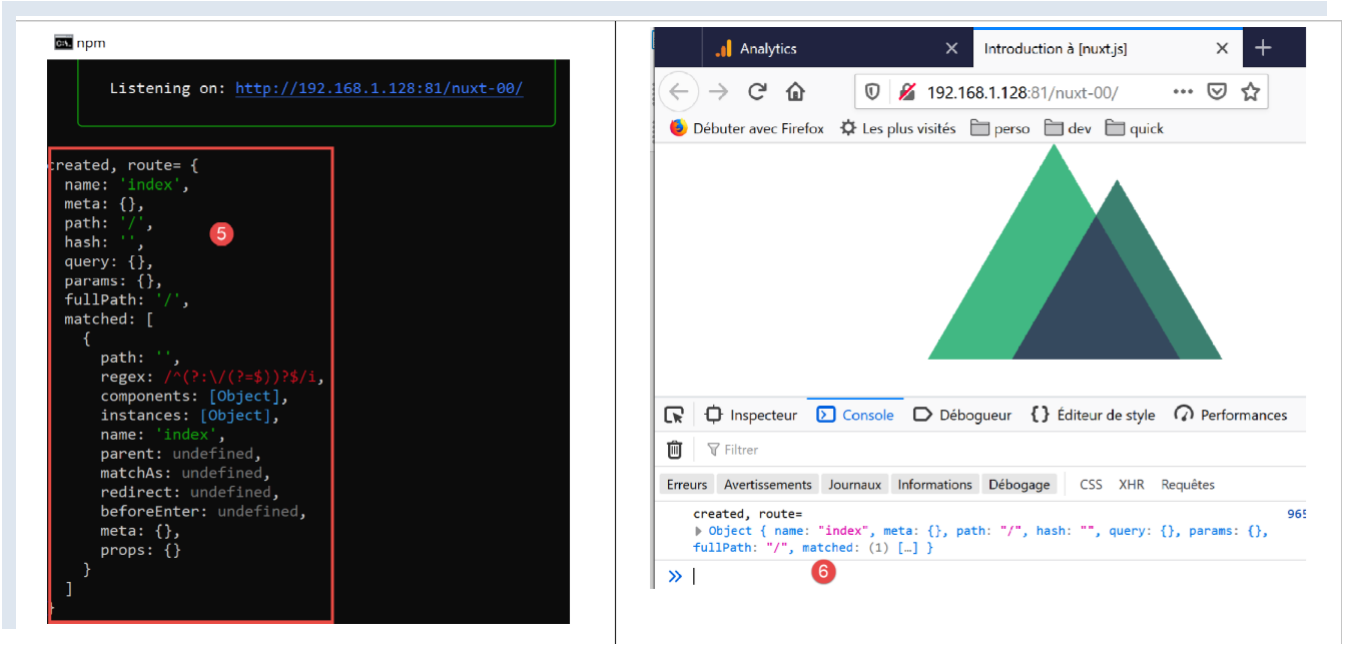
- ligne 5 : on fixe l'URL de base de l'application compilée ;
- lignes 8-14 : on définit le port de service et les adresses réseau écoutées ;

Ceci fait, on ouvre un terminal Laragon et on se positionne dans le dossier contenant la version compilée du projet. On peut ouvrir tout type de terminal mais il faut que l'exécutable **[npm]** soit dans le PATH du terminal. C'est le cas pour le terminal Laragon.

Ceci fait, on tape la commande **[npm run start]** :



En **[3]**, on voit qu'un serveur a été lancé et qu'il écoute à l'URL **[http://192.168.1.128:81/nuxt-00/]**. Maintenant demandons cette URL avec un navigateur **[4]**. On a bien la même chose qu'auparavant. Côté terminal, des logs ont été écrits **[5]**. C'est le log placé dans la méthode **[created]** de la page **[index.vue]** qui a été exécutée par le serveur **[node.js]**.



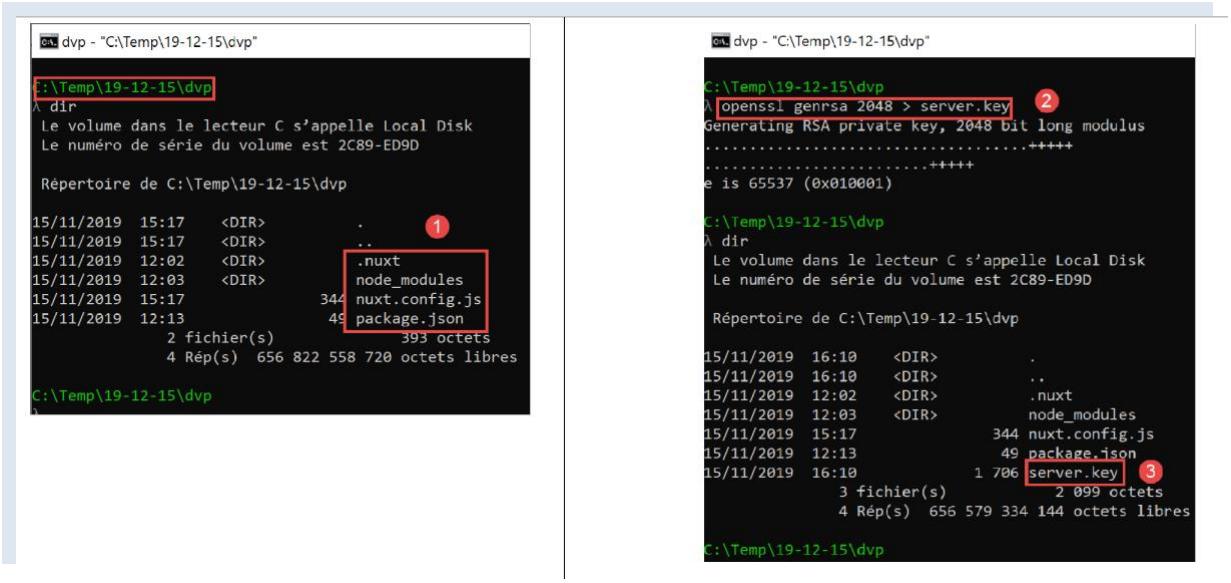
Côté navigateur **[6]**, on retrouve également le log de la méthode **[created]** de la page **[index.vue]** mais exécutée cette fois par le client.

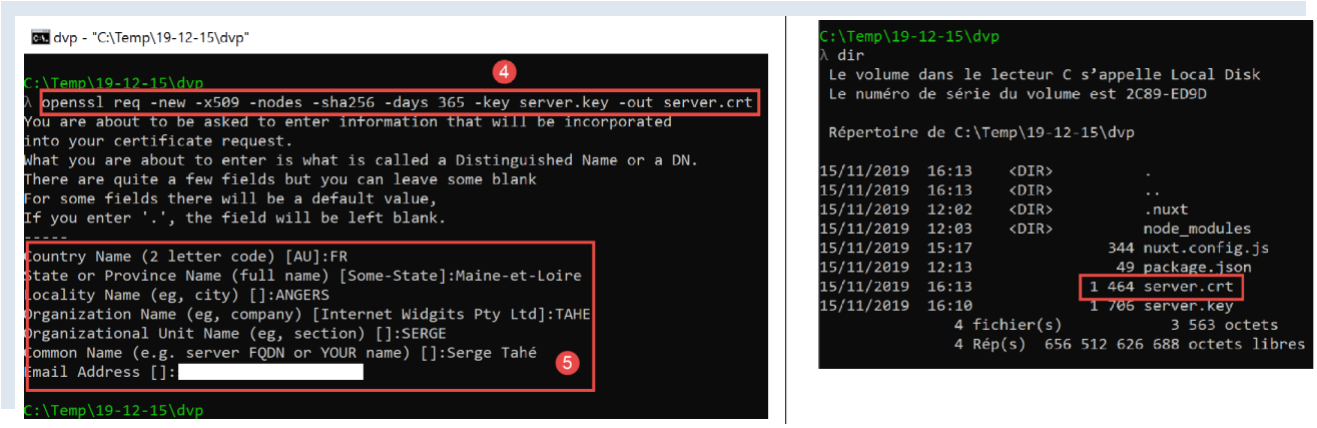
4.3.11 Mise en place d'un serveur sécurisé

Ci-dessus, l'URL de l'application est **[http://192.168.1.128/nuxt-00/]**. On voudrait qu'elle soit **[https://192.168.1.128/nuxt-00/]**. Il nous faut donc construire un serveur sécurisé. Nous montrons comment procéder.

Note : la méthode a été tirée de l'article **[https://stackoverflow.com/questions/56966137/how-to-run-nuxt-npm-run-dev-with-https-in-localhost]**.

Tout d'abord nous créons une clé privée et une clé publique avec **[openssl]**. **[openssl]** est normalement installé en même temps que le serveur Laragon. Du coup, cette commande est disponible dans tout terminal Laragon. Ouvrons donc un terminal Laragon et positionnons-nous sur le dossier de l'application déployée :





- en **[2]**, on tape la commande **[openssl genrsa 2048 > server.key]** ;
- en **[3]**, un fichier **[server.key]** est créé ;
- en **[4]**, on tape la commande **[openssl req -new -x509 -nodes -sha256 -days 365 -key server.key -out server.crt]** ;
- en **[5]**, un fichier **[server.crt]** est créé ;

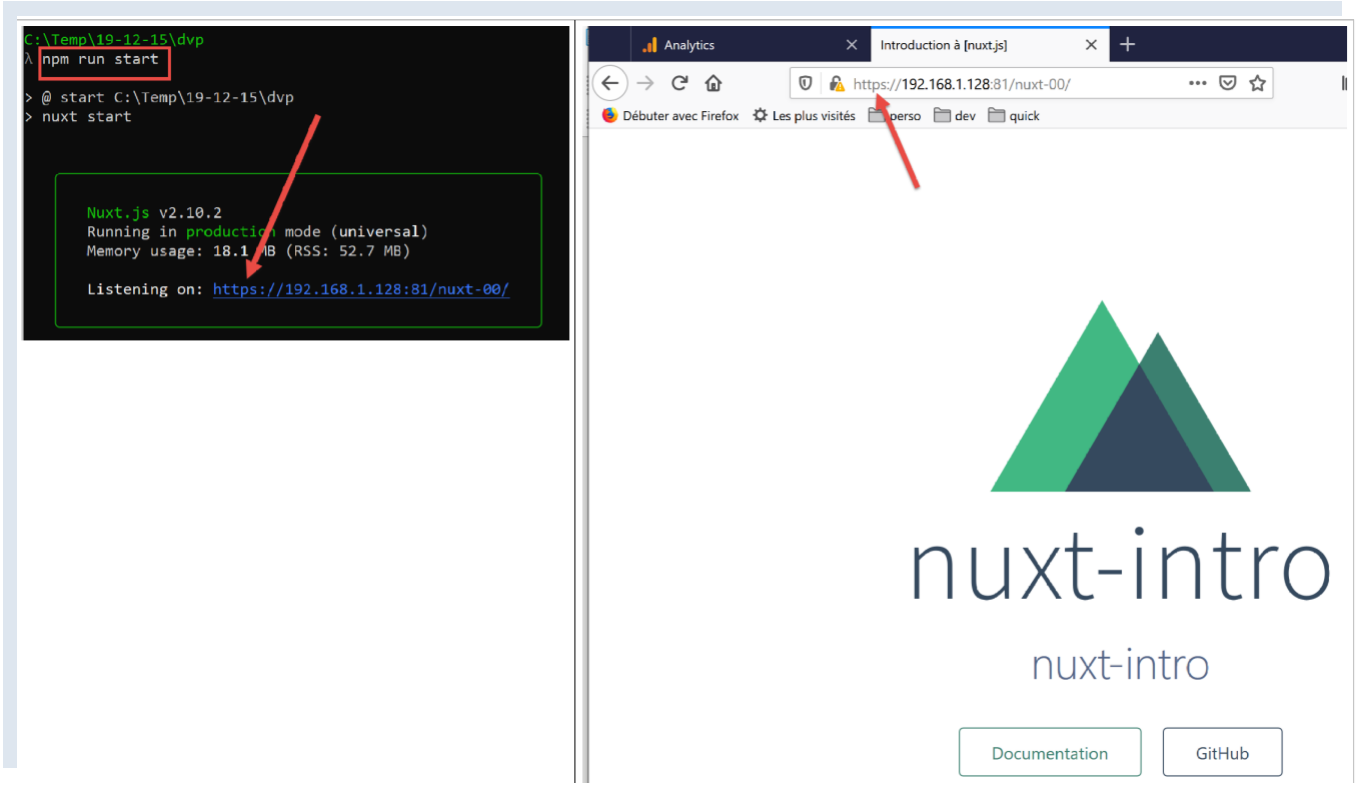
Ces deux fichiers constituent un certificat autosigné. La plupart des navigateurs ne les acceptent qu'après approbation de l'utilisateur ayant demandé la page.

Les fichiers **[server.key, server.crt]** doivent être maintenant utilisés par l'application web. Pour cela le fichier **[nuxt.config.js]** doit être modifié de la façon suivante :

```
1.  import path from 'path'
2.  import fs from 'fs'
3.
4.  export default {
5.    // routeur
6.    router: {
7.      // racine des URL de l'application
8.      base: '/nuxt-00/'
9.    },
10.   // serveur
11.   server: {
12.     // port de service, 3000 par défaut
13.     port: 81,
14.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
15.     // 0.0.0.0 = toutes les adresses réseau de la machine
16.     host: '0.0.0.0',
17.     // certificat autosigné
18.     https: {
19.       key: fs.readFileSync(path.resolve(__dirname, 'server.key')),
20.       cert: fs.readFileSync(path.resolve(__dirname, 'server.crt'))
21.     }
22.   }
23. }
```

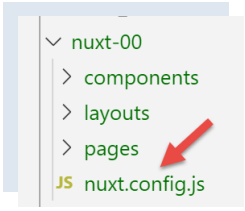
Ce sont les lignes 18-21 qui mettent en place le protocole **[https]**.

Maintenant réexécutons l'application :



4.3.12 Fin du premier exemple

Le premier exemple est désormais terminé. Il nous a appris beaucoup de concepts de **[nuxt]**. Nous allons maintenant développer d'autres exemples que nous placerons dans des dossiers **[nuxt-01, nuxt-02, ...]**. Comme ces exemples utiliseront un fichier **[nuxt.config.js]** différent, nous sauvegarderons dans chacun de ces dossiers, le fichier **[nuxt.config.js]** qui a servi à les exécuter :

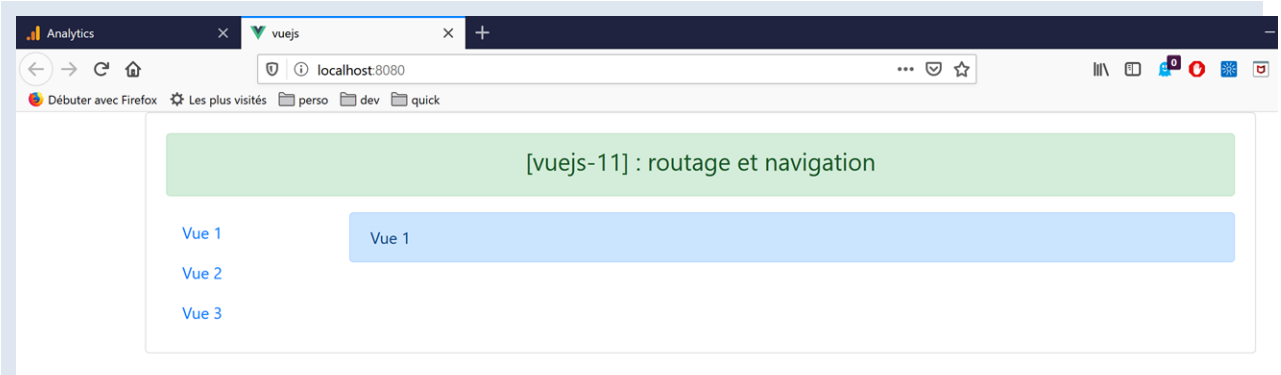


4.4 Exemple [nuxt-01] : routage et navigation

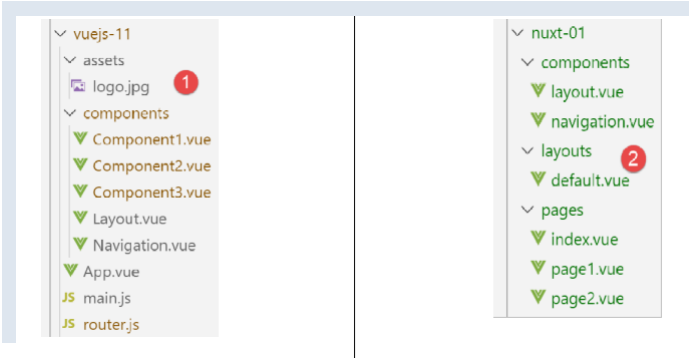
Nous allons construire une série d'exemples simples pour découvrir progressivement le fonctionnement d'une application **[nuxt]**. Nous allons commencer par porter l'application **[vuejs-11]** du document | Introduction au framework **VUE.JS** par l'exemple |, pour découvrir tout d'abord ce qui différencie l'organisation du code d'une application **[nuxt]** de celle du code d'une application **[vue]**.

4.4.1 Arborescence du projet

Le projet **[vuejs-11]** était un projet de navigation entre vues :



L'arborescence du code source du projet **[vuejs-11]** était le suivant :



- **[main.js]** était le script exécuté lors du démarrage de l'application **[vue]** ;
- **[router.js]** fixait les règles de routage ;
- **[App.vue]** était la vue structurante de l'application. Elle organisait la mise en page des différentes vues ;
- **[Component1, Component2, Component3, Layout, Navigation]** étaient les composants utilisés dans les différentes vues de l'application ;

Dans le portage de l'application **[vue] [1]** vers une application **[nuxt] [2]** :

- les scripts exécutés au démarrage de l'application doivent être déclarés dans la clé **[plugins]** du fichier **[nuxt.config.js]**. Par ailleurs, il est possible de séparer les scripts destinés au serveur **[nuxt]** de ceux destinés au client **[nuxt]** ;
- la vue **[App.vue]** doit être installée dans le dossier **[layouts]** et être renommée **[default.vue]** ;
- les composants **[Component1, Component2, Component3]** qui sont les cibles du routage doivent migrer dans le dossier **[pages]**. L'un d'eux, celui qui sert de page d'accueil, doit être renommé **[index.vue]**. Nous avons ici renommé les fichiers :
- **[Component1]** --> **[index]** : affiche le texte **[Home]** ;
- **[Component2]** --> **[page1]** : affiche le texte **[Page 1]** ;
- **[Component3]** --> **[page2]** : affiche le texte **[Page 2]** ;
- **[nuxt]** utilise le contenu du dossier **[pages]** pour générer dynamiquement les routes suivantes :

```
1.  { name :'index', 'path' :'/'}
2.  { name :'page1', 'path' :'/page1'}
3.  { name :'page2', 'path' :'/page2'}
```

Du coup, le fichier **[router.js]** utilisé dans le projet **[vue]** devient inutile dans le projet **[nuxt]**.

Le fichier de configuration **[nuxt.config.js]** sera le suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
```

```
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: { color: '#fff' },
23.   /*
24.    ** Global CSS
25.    */
26.   css: [],
27.   /*
28.    ** Plugins to load before mounting the App
29.    */
30.   plugins: [],
31.   /*
32.    ** Nuxt.js dev-modules
33.    */
34.   buildModules: [
35.     // Doc: https://github.com/nuxt-community/eslint-module
36.     '@nuxtjs/eslint-module'
37.   ],
38.   /*
39.    ** Nuxt.js modules
40.    */
41.   modules: [
42.     // Doc: https://bootstrap-vue.js.org
43.     'bootstrap-vue/nuxt',
44.     // Doc: https://axios.nuxtjs.org/usage
45.     '@nuxtjs/axios'
46.   ],
47.   /*
48.    ** Axios module configuration
49.    ** See https://axios.nuxtjs.org/options
50.    */
51.   axios: {},
52.   /*
53.    ** Build configuration
54.    */
55.   build: {
56.     /*
57.      ** You can extend webpack config here
58.      */
59.     extend(config, ctx) {}
60.   },
61.   // répertoire du code source
62.   srcDir: 'nuxt-01',
63.   // routeur
64.   router: {
65.     // racine des URL de l'application
66.     base: '/nuxt-01/'
67.   },
68.   // serveur
69.   server: {
70.     // port de service, 3000 par défaut
71.     port: 81,
72.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
73.     // 0.0.0.0 = toutes les adresses réseau de la machine
74.     host: '0.0.0.0'
75.   }
76. }
```

- ligne 62 : on indique le dossier qui contient le code source du projet **[dvp]** ;
- ligne 66 : on indique l'URL racine de l'application **[dvp]** (on peut mettre ce qu'on veut) ;
- ligne 43 : notons que la bibliothèque **[bootstrap-vue]** est référencée dans la configuration ;

4.4.2 Portage du fichier [main.js]

Le fichier **[main.js]** du projet **[vuejs-11]** était le suivant :

```
1.  // imports
```

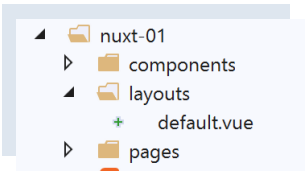
```
2.  import Vue from 'vue'
3.  import App from './App.vue'
4.
5.  // plugins
6.  import BootstrapVue from 'bootstrap-vue'
7.  Vue.use(BootstrapVue);
8.
9.  // bootstrap
10. import 'bootstrap/dist/css/bootstrap.css'
11. import 'bootstrap-vue/dist/bootstrap-vue.css'
12.
13. // routeur
14. import monRouteur from './router'
15.
16. // configuration
17. Vue.config.productionTip = false
18.
19. // instanciation projet [App]
20. new Vue({
21.   name: "app",
22.   // vue principale
23.   render: h => h(App),
24.   // routeur
25.   router: monRouteur,
26. }).$mount('#app')
```

En-dehors des **[imports]**, le code fait les choses suivantes :

- lignes 5-11 : utilisation de la bibliothèque **[bootstrap-vue]**. Ce travail est désormais fait par le module **[bootstrap-vue/nuxt]** de la ligne 43 du fichier de configuration **[nuxt.config.js]** ;
- lignes 14 et 25 : utilisation du fichier de routage **[router.js]**. Ce travail est désormais fait automatiquement par l'application **[nuxt]** à partir de l'arborescence du dossier **[pages]** ;
- lignes 20-26 : instanciation de la vue principale de l'application. Dans une application **[nuxt]**, c'est la vue **[layouts/default.vue]** qui sert de vue principale ;

Le fichier **[main.js]** n'a désormais plus de raison d'être. S'il en avait eu une, on l'aurait déclaré dans la clé **[plugins]** de la ligne 30 du fichier de configuration **[nuxt.config.js]** ;

4.4.3 La vue principale [default.vue]

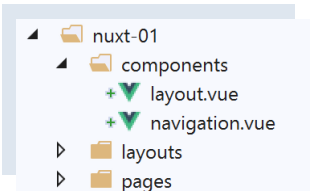


La vue principale **[layouts / default.vue]** est la suivante :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- un message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[nuxt-01] : routage et navigation</h4>
7.        </b-alert>
8.        <!-- la vue courante du routage -->
9.        <nuxt />
10.     </b-card>
11.   </div>
12. </template>
13.
14. <script>
15. export default {
16.   name: 'App'
17. }
18. </script>
```

- ligne 9, dans le projet **[vuejs-11]**, on avait la balise <**router-view** /> au lieu de la balise <**nuxt** /> utilisée ici. Les deux semblent utilisables. Je les ai essayées toutes les deux sans voir de changement. J'ai gardé la balise <**nuxt** /> qui est celle conseillée. Elle affiche la vue courante, ç-à-d la page cible du routage courant ;

4.4.4 Les composants



Par rapport au projet **[vuejs-11]**, les composants **[layout, navigation]** ne changent pas :

[components / layout.vue]

```
1.  <!-- disposition des vues -->
2.  <template>
3.    <!-- ligne -->
4.    <div>
5.      <b-row>
6.        <!-- zone à trois colonnes -->
7.        <b-col v-if="left" cols="2">
8.          <slot name="left" />
9.        </b-col>
10.       <!-- zone à neuf colonnes -->
11.       <b-col v-if="right" cols="10">
12.         <slot name="right" />
13.       </b-col>
14.     </b-row>
15.   </div>
16. </template>
17.
18. <script>
19. export default {
20.   // paramètres
21.   props: {
22.     left: {
23.       type: Boolean
24.     },
25.     right: {
26.       type: Boolean
27.     }
28.   }
29. }
30. </script>
```

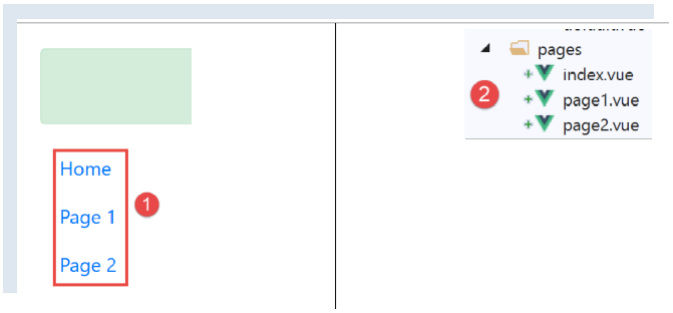
Ce composant sert à structurer les pages de l'application en deux colonnes :

- lignes 7-9 : la colonne de gauche sur 2 colonnes Bootstrap ;
- lignes 11-13 : la colonne de droite sur 10 colonnes Bootstrap ;

[navigation.vue]

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <b-nav-item to="/" exact exact-active-class="active">
5.        Home
6.      </b-nav-item>
7.      <b-nav-item to="/page1" exact exact-active-class="active">
8.        Page 1
9.      </b-nav-item>
10.     <b-nav-item to="/page2" exact exact-active-class="active">
11.       Page 2
12.     </b-nav-item>
13.   </b-nav>
14. </template>
```

Ce composant affiche trois liens de navigation :



Pour savoir quoi mettre comme valeur aux attributs **[to]** des lignes 4, 7 et 10, il faut regarder le dossier **[pages] [2]** :

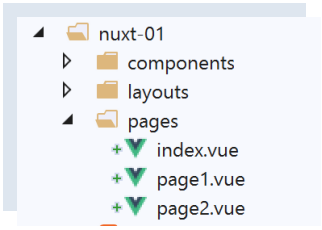
- la page **[index]** aura l'URL **[/]** ;
- la page **[page1]** aura l'URL **[/page1]** ;
- la page **[page2]** aura l'URL **[/page2]** ;

Le composant **[navigation]** peut être également écrit de la façon suivante :

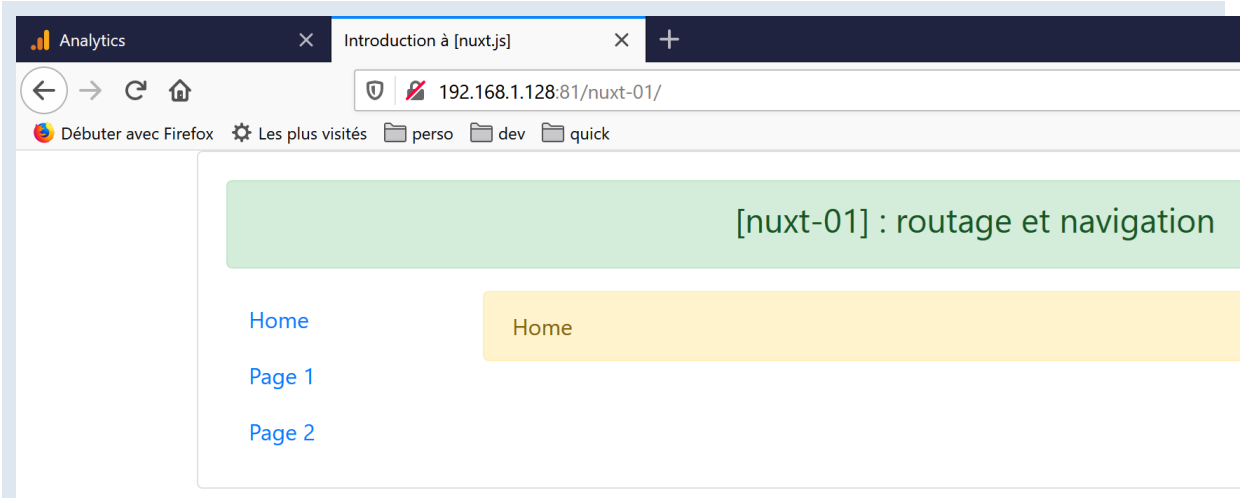
```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <nuxt-link to="/" exact exact-active-class="active">
5.        Home
6.      </nuxt-link>
7.      <nuxt-link to="/page1" exact exact-active-class="active">
8.        Page 1
9.      </nuxt-link>
10.     <nuxt-link to="/page2" exact exact-active-class="active">
11.       Page 2
12.     </nuxt-link>
13.   </b-nav>
14. </template>
```

La balise <**b-nav-item**> est remplacée par la balise <**nuxt-link**> qui désigne un lien de routage. A l'exécution, je n'ai pas vu de grande différence, rien qui pourrait faire pencher la balance vers une balise plutôt que l'autre.

4.4.5 Les pages



La page **[index.vue]** affiche la vue suivante :



Le code de la page est le suivant :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">
8.        Home
9.      </b-alert>
10.   </Layout>
11. </template>
12.
13. <script>
14. /* eslint-disable no-undef */
15. /* eslint-disable no-console */
16. /* eslint-disable nuxt/no-env-in-hooks */
17.
18. import Navigation from '@/components/navigation'
19. import Layout from '@/components/layout'
20.
21. export default {
22.   name: 'Home',
23.   // composants utilisés
24.   components: {
25.     Layout,
26.     Navigation
27.   },
28.   // cycle de vie
29.   beforeCreate(...args) {
30.     console.log('[home beforeCreate]', 'process.server=', process.server, 'process.client=', process.client, "nombre d'a
    rguments=", args.length)
31.   },
32.   created(...args) {
33.     console.log('[home created]', 'process.server=', process.server, 'process.client=', process.client, "nombre d'argumen
    ts=", args.length)
34.   },
35.   beforeMount(...args) {
36.     console.log('[home beforeMount]', 'process.server=', process.server, 'process.client=', process.client, "nombre d'ar
    guments=", args.length)
37.   },
38.   mounted(...args) {
39.     console.log('[home mounted]', 'process.server=', process.server, 'process.client=', process.client, "nombre d'argumen
    ts=", args.length)
40.   }
41. }
42. </script>
```

- ligne 5 : le composant de navigation est placé en colonne de gauche ;
- lignes 7-9 : une alerte est placée dans la colonne de droite ;

Dans la partie <script>, nous mettons du code dans les fonctions du cycle de vie de la page **[beforeCreate, created, beforeMount, beforeMounted]**. Nous voulons savoir lesquelles sont exécutées par le serveur **[]**nuxt et lesquelles par le client **[nuxt]**. On rappelle deux choses :

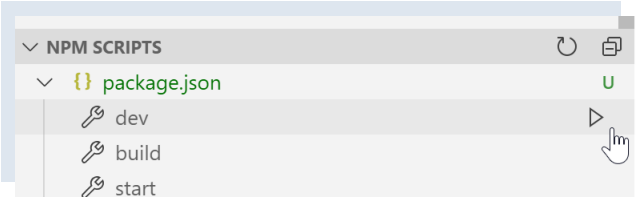
- lorsqu'une page est demandée soit au démarrage de l'application, cas de la page **[index]**, soit manuellement par l'utilisateur qui rafraîchit la page du navigateur ou tape une URL à la main, elle est délivrée d'abord par le serveur **[nuxt]**. Celui-ci interprète le code ci-dessus et exécute le Javascript qu'il contient ;
- lorsque la page envoyée par le serveur **[nuxt]** arrive sur le navigateur, elle arrive avec le code du client **[nuxt]**. Celui-ci interprète de nouveau la page ci-dessus ;
- par des logs, on veut savoir qui fait quoi pour mieux comprendre ce processus ;
- ligne 30 : on utilise dans la fonction un objet global **[process]** qui existe aussi bien sur le serveur que sur le client :
  - **[process.server]** est vrai si le code est exécuté par le serveur, faux sinon ;
  - **[process.client]** est vrai si le code est exécuté par le client, faux sinon ;
- parce que la variable **[process]** est non déclarée dans le code, on est obligés de mettre la ligne 14 pour **[eslint]**. La ligne **[16]** est nécessaire parce que sinon **[eslint]** déclare un autre type d'erreur à cause de la variable **[process]**. La ligne 15 est elle nécessaire pour permettre l'utilisation de **[console]** dans les fonctions du cycle de vie ;
- ligne 29 : on veut savoir également si les fonctions du cycle de vie reçoivent des arguments. On va découvrir en effet que **[nuxt]** transmet des informations à certaines fonctions. On veut savoir si les fonctions du cycle de vie en font partie ;
- on répète le même code pour les quatre fonctions ;

4.4.6 Le fichier [nuxt.config.js]

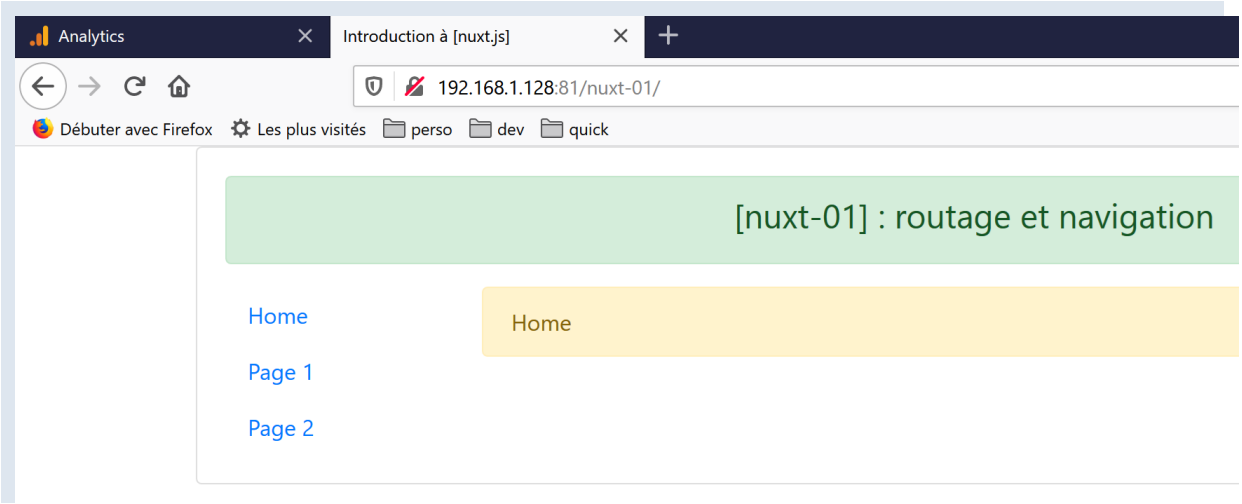
C'est lui qui contrôle l'exécution du projet. Il a été décrit |ici|.

4.4.7 Exécution du projet

Nous exécutons le projet :



La page affichée est la suivante :



Une fois installée sur le navigateur, l'application **[nuxt]** devient une application **[vue]** classique. Nous ne commenterons donc pas le fonctionnement client de l'application **[nuxt-01]**. Cela a été fait dans le projet **[vuejs-11]** du document |Introduction au framework **VUE.JS** par l'exemple|.

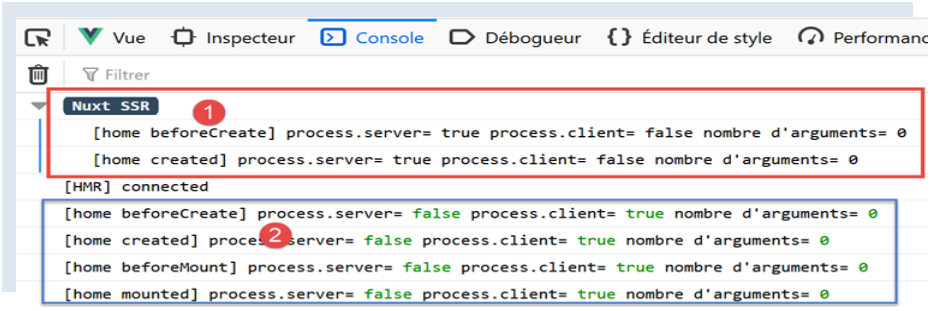
L'application **[nuxt]** ne diffère de l'application **[vue]** qu'à deux moments :

- le démarrage initial de l'application qui fournit la page d'accueil ;
- à chaque fois que l'utilisateur provoque d'une manière ou une autre le rafraîchissement du navigateur ;

Dans ces deux cas :

- la page demandée est fournie par le serveur ;
- la page reçue est traitée par le client ;

Regardons les logs du démarrage de l'application (F12 sur le navigateur) :



- en **[1]**, les logs du serveur (process.server=true). Ils apparaissent précédés de la mention **[Nuxt SSR]** (SSR= Server Side Rendered) ;
- en **[2]**, les logs du client sur le navigateur (process.client=true) ;

De ces logs, on peut déduire que :

- le **serveur** exécute les fonctions **[beforeCreate, created]** du cycle de vie ;
- le **client** exécute les fonctions **[beforeCreate, created, beforeMount, mounted]** du cycle de vie ;
- le serveur a traité la page **avant** le client ;
- dans les deux cas, aucune des fonctions exécutées ne reçoit d'arguments ;

Maintenant regardons le code source de la page reçue (option **[Code source de la page]** dans le navigateur) :

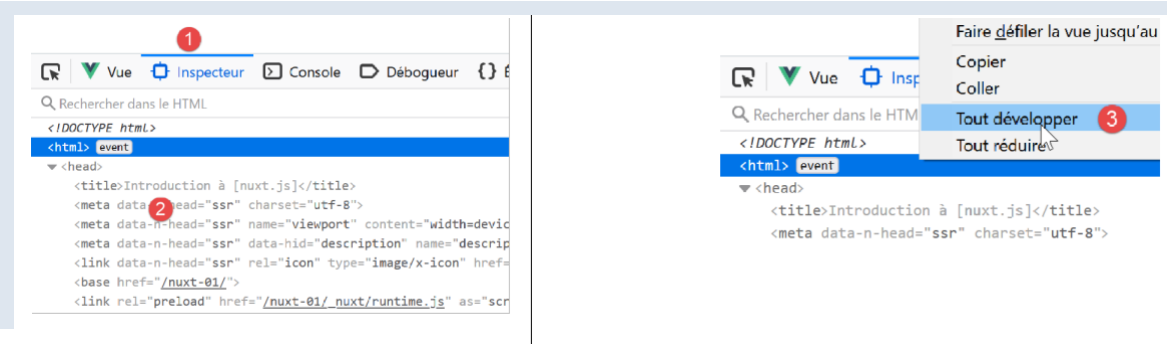
```
<!doctype html>
<html data-n-head-ssr>
<head>
  <title>Introduction à [nuxt.js]</title>
  <meta data-n-head="ssr" charset="utf-8">
  <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
  <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata middleware plugins store">
  <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
  <base href="/nuxt-01/">
  ....
  <link rel="preload" href="/nuxt-01/_nuxt/runtime.js" as="script">
  <link rel="preload" href="/nuxt-01/_nuxt/commons.app.js" as="script">
  <link rel="preload" href="/nuxt-01/_nuxt/vendors.app.js" as="script">
  <link rel="preload" href="/nuxt-01/_nuxt/app.js" as="script">
</head>
<body>
  <div data-server-rendered="true" id="__nuxt">
    <div id="__layout">
      <div class="container">
        <div class="card">
          <div class="card-body">
            <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-success">
              <h4>[nuxt-01] : routage et navigation</h4>
            </div>
            <div>
              <div class="row">
                <div class="col-2">
                  <ul class="nav flex-column">
                    <li class="nav-item">
                      <a href="/nuxt-01/" target="_self" class="nav-link active nuxt-link-active">
                        Home
                      </a>
                    </li>
                    <li class="nav-item">
                      <a href="/nuxt-01/page1" target="_self" class="nav-link">
                        Page 1
                      </a>
                    </li>
                    <li class="nav-item">
                      <a href="/nuxt-01/page2" target="_self" class="nav-link">
                        Page 2
```

```
42.                     </a>
43.                   </li>
44.                 </ul>
45.               </div> <div class="col-10">
46.                 <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-warning">
47.                   Home
48.                 </div>
49.               </div>
50.             </div>
51.           </div>
52.         </div>
53.       </div>
54.     </div>
55.   </div>
56.   <script>
57.     window.__NUXT__ = (function (a, b, c, d, e, f, g, h, i, j) {
58.       return {
59.         layout: "default", data: [{}], error: null, serverRendered: true,
60.         logs: [
61.           { date: new Date(1574069600078), args: [a, b, c, d, e, f, g, "(repeated 1 times)"], type: h,
    level: i, tag: j },
62.           { date: new Date(1574070938091), args: [a, b, c, d, e, f, g], type: h, level: i, tag: j }
63.         ]
64.       }
65.     }("[home beforeCreate]", "process.server=", "true", "process.client=", "false", "nombre d'arguments=",
    "0", "log", 2, ""));
66.   </script>
67.   <script src="/nuxt-01/_nuxt/runtime.js" defer></script>
68.   <script src="/nuxt-01/_nuxt/commons.app.js" defer></script>
69.   <script src="/nuxt-01/_nuxt/vendors.app.js" defer></script>
70.   <script src="/nuxt-01/_nuxt/app.js" defer></script>
71. </body>
72. </html>
```

Commentaires

- la première chose qui peut être remarquée est que le code HTML **reçu** reflète correctement ce que voit l'utilisateur. Ce n'était pas le cas des applications **[vue]** pour lesquelles le code source affiché était le code source d'un fichier HTML quasi vide. C'était ce qu'avait reçu le navigateur. Ensuite le client **[vue]** prenait la main et construisait la page attendue par l'utilisateur. Il fallait alors aller dans l'onglet **[inspecteur]** des outils de développement du navigateur (F12) pour découvrir le code HTML de la page affichée ;
- lignes 57-67 : c'est le script qui a affiché les logs tagués **[Nuxt SSR]**. Ces logs ont été produits côté serveur et les résultats ont été embarqués dans un script inclus dans la page envoyée ;
- lignes 68-71 : les scripts qui forment le client exécuté côté navigateur ;

Les scripts des lignes 68-71 sont exécutés et transforment la page reçue. Pour connaître la page finalement affichée pour l'utilisateur, il faut aller dans l'onglet **[inspecteur]** des outils de développement du navigateur (F12) :



Lorsqu'on développe la balise <html> **[3]**, on a le contenu suivant :

```
1.  <head>
2.    <title>Introduction à [nuxt.js]</title>
3.    <meta data-n-head="ssr" charset="utf-8">
4.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
5.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata
    middleware plugins store">
6.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
```

```
7.    <base href="/nuxt-01/">
8.    ...
9.    <link rel="preload" href="/nuxt-01/_nuxt/runtime.js" as="script">
10.   <link rel="preload" href="/nuxt-01/_nuxt/commons.app.js" as="script">
11.   <link rel="preload" href="/nuxt-01/_nuxt/vendors.app.js" as="script">
12.   <link rel="preload" href="/nuxt-01/_nuxt/app.js" as="script">
13.
14.   <script charset="utf-8" src="/nuxt-01/_nuxt/pages_index.js"></script>
15.   <script charset="utf-8" src="/nuxt-01/_nuxt/pages_page1.js"></script>
16.   <script charset="utf-8" src="/nuxt-01/_nuxt/pages_page2.js"></script>
17. </head>
18. <body>
19.   <div id="__nuxt">
20.     <div id="__layout">
21.       <div class="container">
22.         <div class="card">
23.           <div class="card-body">
24.             <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-success"
    align="center">
25.               <h4>[nuxt-01] : routage et navigation</h4>
26.             </div>
27.             <div>
28.               <div class="row">
29.                 <div class="col-2">
30.                   <ul class="nav flex-column">
31.                     <li class="nav-item">
32.                       <a href="/nuxt-01/" target="_self" class="nav-link active nuxt-link-active">
33.                         Home
34.                       </a>
35.                     </li>
36.                     <li class="nav-item">
37.                       <a href="/nuxt-01/page1" target="_self" class="nav-link">
38.                         Page 1
39.                       </a>
40.                     </li>
41.                     <li class="nav-item">
42.                       <a href="/nuxt-01/page2" target="_self" class="nav-link">
43.                         Page 2
44.                       </a>
45.                     </li>
46.                   </ul>
47.                 </div>
48.                 <div class="col-10">
49.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-warning">
50.                     Home
51.                   </div>
52.                 </div>
53.               </div>
54.             </div>
55.           </div>
56.         </div>
57.       </div>
58.     </div>
59.   </div>
60.   <script>
61.     window.__NUXT__ = (function (a, b, c, d, e, f, g, h, i) {
62.       return {
63.         layout: "default", data: [{}], error: null, serverRendered: true,
64.         logs: [
65.           { date: new Date(1574068674481), args: ["[home beforeCreate]", a, b, c, d, e, f], type: g, level:
    h, tag: i },
66.           { date: new Date(1574068674482), args: ["[home created]", a, b, c, d, e, f], type: g, level: h,
    tag: i }
67.         ]
68.       }
69.     }("process.server=", "true", "process.client=", "false", "nombre d'arguments=", "0", "log", 2, ""));
70.   </script>
71.   <script src="/nuxt-01/_nuxt/runtime.js" defer=""></script>
72.   <script src="/nuxt-01/_nuxt/commons.app.js" defer=""></script>
73.   <script src="/nuxt-01/_nuxt/vendors.app.js" defer=""></script>
74.   <script src="/nuxt-01/_nuxt/app.js" defer=""></script>
75.
76.   <iframe id="mc-sidebar-container" ...></iframe>
77.   <iframe id="mc-topbar-container"...>  </iframe>
78.   <iframe id="mc-toast-container" ...></iframe>
79.   <iframe id="mc-download-overlay-container"...></iframe>
80. </body>
```

Commentaires

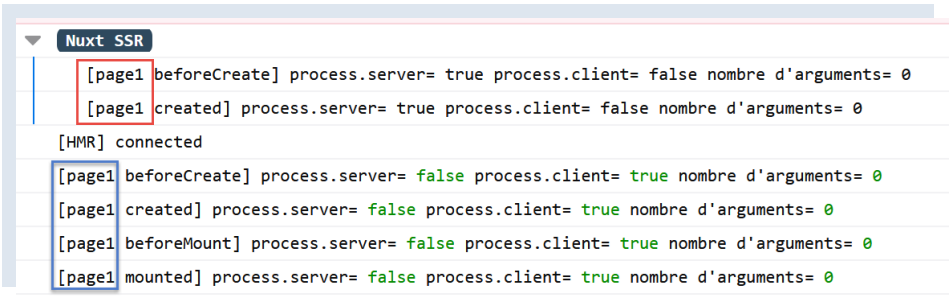
- à première vue, la page affichée lignes 19-59, semble être la même que la page reçue ;
- lignes 14-16 : trois nouveaux scripts apparaissent, un pour chacune des pages de l'application ;
- lignes 76-79 : quatre **[iframe]** apparaissent ;

Lignes 33, 37 et 42, les liens posent problème. Ils semblent être des liens normaux qui lorsqu'on les clique vont faire une requête vers le serveur. Or à l'exécution, on voit que ce n'est pas vrai : il n'y a pas de requête vers le serveur. Pour comprendre pourquoi, il faut retourner dans l'onglet **[inspecteur]** du navigateur :

On voit qu'en **[1, 2]** des événements ont été attachés aux liens. Ce sont les scripts des lignes 71-74 qui ont attaché des gestionnaires d'événements aux liens. Donc :

- la page affichée par le client est visuellement identique à celle envoyée par le serveur ;
- un comportement dynamique a été ajouté à la page par le client ;

Maintenant demandons la page **[page1]** en tapant l'URL à la main **[http://192.168.1.128:81/nuxt-01/page1]**. Les logs deviennent les suivants :



On obtient les mêmes résultats que pour la page **[index]** mais pour **[page1]**. Le code source de la page reçue est lui le suivant :

```
1.  <body>
2.    <div data-server-rendered="true" id="__nuxt">
3.      <div id="__layout">
4.        <div class="container">
5.          <div class="card">
6.            <div class="card-body">
7.              <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-
    success"><h4>[nuxt-01] : routage et navigation</h4></div> <div>
8.                <div class="row">
9.                  <div class="col-2">
10.                   <ul class="nav flex-column">
11.                     <li class="nav-item">
12.                       <a href="/nuxt-01/" target="_self" class="nav-link">
13.                         Home
14.                       </a>
15.                     </li>
16.                     <li class="nav-item">
17.                       <a href="/nuxt-01/page1" target="_self" class="nav-link active nuxt-link-active">
```

```
18.                       Page 1
19.                     </a>
20.                   </li>
21.                   <li class="nav-item">
22.                     <a href="/nuxt-01/page2" target="_self" class="nav-link">
23.                       Page 2
24.                     </a>
25.                   </li>
26.                 </ul>
27.               </div> <div class="col-10">
28.                 <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-primary">
29.
30.                   Page 1
31.                 </div>
32.               </div>
33.             </div>
34.           </div>
35.         </div>
36.       </div>
37.     </div>
38.   </div>
39.   <script>window.__NUXT__ = { layout: "default", data: [{}], error: null, serverRendered: true, logs:
    [{ date: new Date(1573917721122), args: ["[page1 beforeCreate]", "process.server=", "true",
    "process.client=", "false", "nombre d'arguments=", "0"], type: "log", level: 2, tag: "" }] };</script>
41.   <script src="/nuxt-01/_nuxt/runtime.js" defer></script>
42.   <script src="/nuxt-01/_nuxt/commons.app.js" defer></script>
43.   <script src="/nuxt-01/_nuxt/vendors.app.js" defer></script>
44.   <script src="/nuxt-01/_nuxt/app.js" defer></script>
45. </body>
```

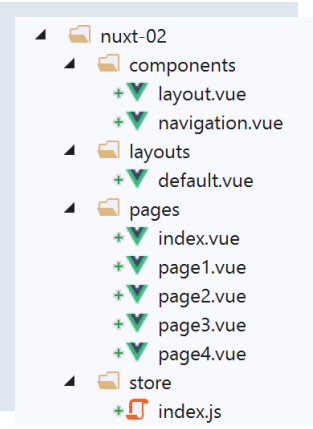
On obtient le même type de page que la page **[index]** mais avec l'alerte de la vue **[Page 1]** (ligne 30). Lignes 41-44, le code du client a été renvoyé avec la page. Au final, demander une URL à la main est **identique** à redémarrer l'application. Simplement la page affichée n'est pas forcément la page d'accueil, c'est celle qui a été demandée. Une fois la page reçue, c'est le client qui prend la main. Le serveur ne sera plus sollicité à moins que l'utilisateur n'en décide autrement.

4.5 Exemple [nuxt-02] : pages serveur et client

Dans ce projet, nous montrons :

- que la page construite par le client peut être visuellement différente de celle reçue du serveur. On a alors un changement rapide de page, perceptible par l'utilisateur, et qui est donc nuisible à l'ergonomie de l'application. C'est donc une option à éviter ;
- une solution pour que la page client recrée la même page que celle envoyée par le serveur ;

Le projet **[nuxt-02]** est obtenue initialement par recopie du projet **[nuxt-01]**.



Un dossier **[store]** est ajouté au projet ainsi que deux nouvelles pages. Nous y reviendrons.

4.5.1 La page [index]

4.5.1.1 Le code de la page

Le code de la page **[index]** devient le suivant :

```
<!-- page principale -->
<template>
  <Layout :left="true" :right="true">
    <!-- navigation -->
    <Navigation slot="left" />
    <!-- message-->
    <b-alert slot="right" show variant="warning"> Home - value= {{ value }} </b-alert>
  </Layout>
</template>

<script>
/* eslint-disable no-undef */
/* eslint-disable no-console */
/* eslint-disable nuxt/no-env-in-hooks */

import Navigation from '@/components/navigation'
import Layout from '@/components/layout'

export default {
  name: 'Home',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  data() {
    return {
      value: 0
    }
  },
  // cycle de vie
  beforeCreate() {
    // client et serveur
    console.log('[home beforeCreate]')
  },
  created() {
    // client et serveur
    console.log('[home created]')
    // serveur seulement
    if (process.server) {
      this.value = 10
    }
    // client et serveur
    console.log('value=', this.value)
  },
  beforeMount() {
    // client seulement
    console.log('[home beforeMount]')
  },
  mounted() {
    // client seulement
    console.log('[home mounted]')
  }
}
</script>
```

Commentaires

- ligne 7 : la page **[index]** va afficher la valeur de sa propriété **[value]** (ligne 28) ;
- lignes 36-45 : il faut se rappeler ici que la fonction **[created]** est exécutée à la fois côté serveur et côté client. Lignes 40-42, le serveur fera passer à 10 la valeur de la propriété **[value]**. Le client, lui, ne touche pas à cette valeur. On veut simplement savoir si cette valeur est conservée par le client. On va découvrir que non ;

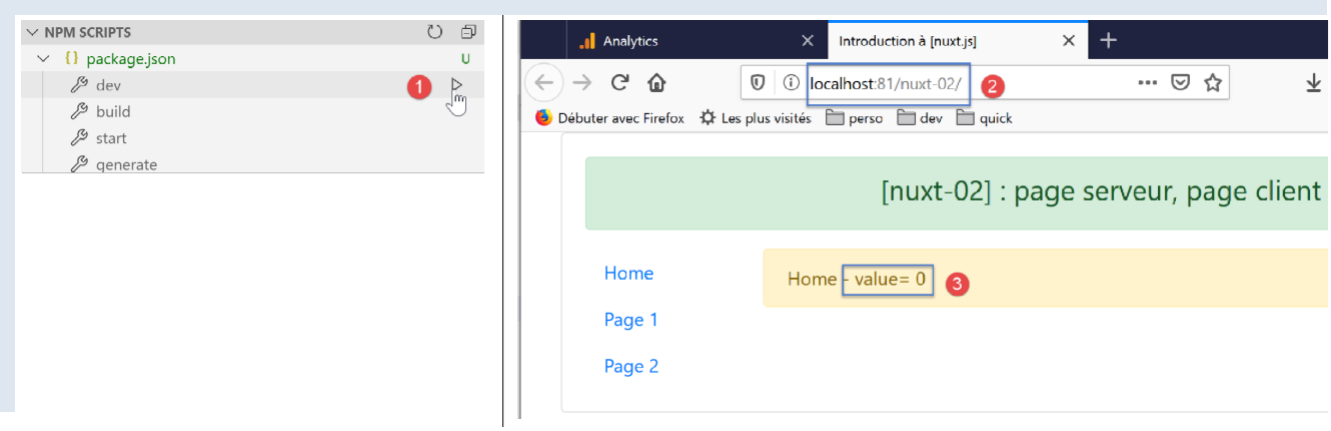
4.5.1.2 Exécution

Nous modifions le fichier **[/nuxt.config.js]** pour exécuter le projet **[nuxt-02]** :

```
…
  // répertoire du code source
  srcDir: 'nuxt-02',
  // routeur
  router: {
    // racine des URL de l'application
    base: '/nuxt-02/'
  },
```

```
9.    // serveur
10.   server: {
11.     // port de service, 3000 par défaut
12.     port: 81,
13.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
14.     // 0.0.0.0 = toutes les adresses réseau de la machine
15.     host: 'localhost'
16.   }
17. …
```

Nous exécutons le projet **[1]** :



La page **[index]** est alors affichée **[2-3]**. Elle affiche la valeur **[10]** pendant quelques instants puis affiche la valeur **[0]**. Que s'est-il passé ?

étape 1

C'est le serveur qui s'exécute le premier. Il exécute le code de la page **[index]** :

```
1.  export default {
2.    name: 'Home',
3.    // composants utilisés
4.    components: {
5.      Layout,
6.      Navigation
7.    },
8.    data() {
9.      return {
10.       value: 0
11.     }
12.   },
13.   // cycle de vie
14.   beforeCreate() {
15.     // client et serveur
16.     console.log('[home beforeCreate]')
17.   },
18.   created() {
19.     // client et serveur
20.     console.log('[home created]')
21.     // serveur seulement
22.     if (process.server) {
23.       this.value = 10
24.     }
25.     // client et serveur
26.     console.log('value=', this.value)
27.   },
28.   beforeMount() {
29.     // client seulement
30.     console.log('[home beforeMount]')
31.   },
32.   mounted() {
33.     // client seulement
34.     console.log('[home mounted]')
35.   }
36. }
```

- à cause de la ligne 23, la propriété **[value]** de la ligne 10 prend la valeur 10 ;

On peut le vérifier en regardant le code source de la page reçue par le navigateur (option **[code source]** dans le navigateur) :

```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata
    middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-02/">
10.   <link rel="preload" href="/nuxt-02/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-02/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-02/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-02/_nuxt/app.js" as="script">
14.   ....
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-
    success">
23.               <h4>[nuxt-02] : page serveur, page client</h4>
24.             </div> <div>
25.               <div class="row">
26.                 <div class="col-2">
27.                   <ul class="nav flex-column">
28.                     <li class="nav-item">
29.                       <a href="/nuxt-02/" target="_self" class="nav-link active nuxt-link-active">
30.                         Home
31.                       </a>
32.                     </li>
33.                     <li class="nav-item">
34.                       <a href="/nuxt-02/page1" target="_self" class="nav-link">
35.                         Page 1
36.                       </a>
37.                     </li>
38.                     <li class="nav-item">
39.                       <a href="/nuxt-02/page2" target="_self" class="nav-link">
40.                         Page 2
41.                       </a>
42.                     </li>
43.                   </ul>
44.                 </div> <div class="col-10">
45.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-warning">
46.                     Home - value= 10
47.                   </div>
48.                 </div>
49.               </div>
50.             </div>
51.           </div>
52.         </div>
53.       </div>
54.     </div>
55.   </div>
56.   <script>window.__NUXT__ = ....;</script>
57.   <script src="/nuxt-02/_nuxt/runtime.js" defer></script>
58.   <script src="/nuxt-02/_nuxt/commons.app.js" defer></script>
59.   <script src="/nuxt-02/_nuxt/vendors.app.js" defer></script>
60.   <script src="/nuxt-02/_nuxt/app.js" defer></script>
61. </body>
62. </html>
```

- ligne 46 : dans la page reçue, **[value]** avait la valeur 10 ;

étape 2

On sait qu'après réception de la page, les scripts des lignes 57-60 prennent la main et transforment le comportement de la page reçue, voire les informations affichées comme ici. Ces scripts forment le client qui lui aussi exécute le code de la page **[index]**, le même code que le serveur :

```
1.  export default {
2.    name: 'Home',
3.    // composants utilisés
4.    components: {
5.      Layout,
6.      Navigation
7.    },
8.    data() {
9.      return {
10.       value: 0
11.     }
12.   },
13.   // cycle de vie
14.   beforeCreate() {
15.     // client et serveur
16.     console.log('[home beforeCreate]')
17.   },
18.   created() {
19.     // client et serveur
20.     console.log('[home created]')
21.     // serveur seulement
22.     if (process.server) {
23.       this.value = 10
24.     }
25.     // client et serveur
26.     console.log('value=', this.value)
27.   },
28.   beforeMount() {
29.     // client seulement
30.     console.log('[home beforeMount]')
31.   },
32.   mounted() {
33.     // client seulement
34.     console.log('[home mounted]')
35.   }
36. }
```

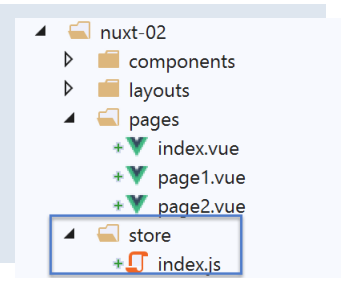
- pour comprendre ce qui se passe, il faut comprendre que le client **[nuxt]** n'exécutera pas les lignes 22-24 (process.server=false) ;
- dans une application **[vue]** classique la propriété **[value]** de la ligne 10 reste à 0. C'est pourquoi, une fois que le client est passé sur la page reçue, la valeur affichée devient **[0]** ;

La valeur générée par le serveur **[nuxt]** pour la propriété **[value]** n'a servi à rien.

4.5.2 La page [page1]

4.5.2.1 Le store [Vuex]

Nous avons ajouté un dossier **[store]** au projet **[nuxt-02]** :



La présence de ce dossier fait qu'automatiquement **[nuxt]** va implémenter un store **[Vuex]**. C'est le fichier **[index.js]** qui implémente ce store. Ici, le fichier **[index.js]** est le suivant :

```
1.  export const state = () => ({
2.    counter: 0
3.  })
4.
5.  export const mutations = {
```

```
6.    increment(state, inc) {
7.      state.counter += inc
8.    }
9.  }
```

[nuxt] implémente un store **[Vuex]** à partir du contenu de **[index.js]** :

- lignes 1-3 : définition de l'état **[state]** du store. Cet état est retourné par une **fonction**. Ici, l'état n'a qu'une propriété, le compteur de la ligne 2. La fonction exportée doit s'appeler **[state]** ;
- lignes 5-9 : les opération possibles sur l'état du store. On les appelle des **[mutations]**. Ici, la mutation **[increment]** permet d'incrémenter la propriété **[counter]** d'une quantité **[inc]**. L'objet exporté doit s'appeler **[mutations]** ;

Le **[store]** Vuex implémenté par **[nuxt]** est disponible à différents endroits. Dans les vues, il est disponible dans la propriété **[this.$store]**.

4.5.2.2 Le code de la page

Comme la page **[index]**, la page **[page1]** va afficher une valeur, celle du compteur du store Vuex :

```
1.  <!-- page page1 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="primary"> Page 1 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13.
14. import Navigation from '@/components/navigation'
15. import Layout from '@/components/layout'
16.
17. export default {
18.   name: 'Page1',
19.   // composants utilisés
20.   components: {
21.     Layout,
22.     Navigation
23.   },
24.   data() {
25.     return {
26.       value: 0
27.     }
28.   },
29.   // cycle de vie
30.   beforeCreate() {
31.     // client et serveur
32.     console.log('[page1 beforeCreate]')
33.   },
34.   created() {
35.     // client et serveur
36.     console.log('[page1 created]')
37.     // serveur seulement
38.     if (process.server) {
39.       this.$store.commit('increment', 25)
40.     }
41.     // client et serveur
42.     this.value = this.$store.state.counter
43.     console.log('value=', this.value)
44.   },
45.   beforeMount() {
46.     // client seulement
47.     console.log('[page1 beforeMount]')
48.   },
49.   mounted() {
50.     // client seulement
51.     console.log('[page1 mounted]')
52.   }
53. }
54. </script>
```

Commentaires

- lignes 38-40 : le serveur va incrémenter le compteur de 25 ;
- ligne 42 : aussi bien le serveur que le client vont afficher la valeur du compteur ;
- ligne 7 : la valeur du compteur est affichée ;

En lisant ce code, il faut comprendre deux choses :

- le code exécuté est le même pour le serveur que le client ;
- l'objet **[this]** n'est lui pas le même : il y a une version **[this]** côté serveur et une autre côté **[client]** ;

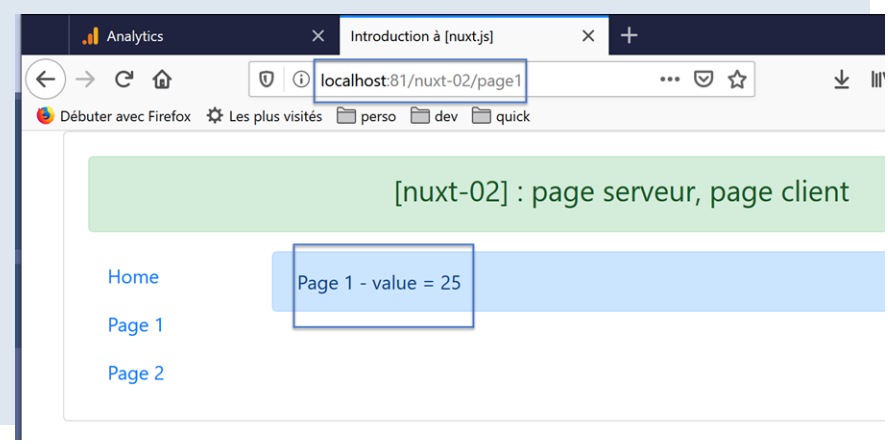
Nous cherchons à savoir si le **[this.$store]** du serveur est le même que le **[this.$store]** du client. Comme c'est le serveur qui s'exécute en premier (au démarrage de l'application), cela revient à se poser la question : est-ce que le **[store]** initialisé par le serveur est transmis au client ?

4.5.2.3 Exécution

On exécute le projet **[nuxt-02]** et on tape **[localhost:81/nuxt-02/page1]** à la main pour que le serveur soit sollicité. Comme au démarrage pour la page **[index]** :

- le serveur exécute la page **[page1.vue]** ;
- envoie la page générée au navigateur. Celle-ci est affichée ;
- les scripts client embarqués dans la page envoyée prennent la main et exécutent à nouveau la page **[page1.vue]** ;
- la page affichée est alors modifiée ;

Le résultat final est le suivant :



Cette fois-ci, la valeur affichée est bien celle fixée par le serveur et visuellement, on ne voit pas la page 'tressauter' à cause d'un changement par le client de la valeur affichée par le serveur. Cette fois-ci que s'est-il passé ?

Le serveur a exécuté la page **[page1]** suivante :

```
1.  ...
2.
3.  <script>
4.  /* eslint-disable no-console */
5.
6.  import Navigation from '@/components/navigation'
7.  import Layout from '@/components/layout'
8.
9.  export default {
10.   name: 'Page1',
11.   // composants utilisés
12.   components: {
13.     Layout,
14.     Navigation
15.   },
16.   data() {
17.     return {
18.       value: 0
19.     }
20.   },
```

```
21.   // cycle de vie
22.   beforeCreate() {
23.     // client et serveur
24.     console.log('[page1 beforeCreate]')
25.   },
26.   created() {
27.     // client et serveur
28.     console.log('[page1 created]')
29.     // serveur seulement
30.     if (process.server) {
31.       this.$store.commit('increment', 25)
32.     }
33.     // client et serveur
34.     this.value = this.$store.state.counter
35.     console.log('value=', this.value)
36.   },
37.   beforeMount() {
38.     // client seulement
39.     console.log('[page1 beforeMount]')
40.   },
41.   mounted() {
42.     // client seulement
43.     console.log('[page1 mounted]')
44.   }
45. }
46. </script>
```

- les lignes 30-32 ont été exécutées sans erreur. Ce qui signifie que côté serveur également, **[this.$store]** désigne le store **[Vuex]**. La ligne 31 a fait passer le compteur du store à 25 ;
- ceci fait, la page a été envoyée au client ;

Si on regarde la page reçue par le client, on trouve les éléments suivants :

```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata
    middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-02/">
10.   <link rel="preload" href="/nuxt-02/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-02/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-02/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-02/_nuxt/app.js" as="script">
14.   ...
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-
    success">
23.               <h4>[nuxt-02] : page serveur, page client</h4>
24.             </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
29.                     <li class="nav-item">
30.                       <a href="/nuxt-02/" target="_self" class="nav-link">
31.                         Home
32.                       </a>
33.                     </li>
34.                     <li class="nav-item">
35.                       <a href="/nuxt-02/page1" target="_self" class="nav-link active nuxt-link-active">
36.                         Page 1
37.                       </a>
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-02/page2" target="_self" class="nav-link">
41.                         Page 2
```

```
42.                     </a>
43.                   </li>
44.                 </ul>
45.               </div> <div class="col-10">
46.                 <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-primary">
47.                   Page 1 - value = 25
48.                 </div>
49.               </div>
50.             </div>
51.           </div>
52.         </div>
53.       </div>
54.     </div>
55.   </div>
56. </div>
57. <script>
58. window.__NUXT__ = (function (a, b, c) {
59.     return {
60.       layout: "default", data: [{}], error: null, state: { counter: 25 }, serverRendered: true,
61.       logs: [
62.         { date: new Date(1574085336802), args: ["[home beforeCreate]"], type: a, level: b, tag: c },
63.         { date: new Date(1574085336839), args: ["[home created]"], type: a, level: b, tag: c },
64.         { date: new Date(1574085336869), args: ["value=", "25"], type: a, level: b, tag: c }
65.       ]
66.     }
67.   }("log", 2, ""));</script>
68. <script src="/nuxt-02/_nuxt/runtime.js" defer></script>
69. <script src="/nuxt-02/_nuxt/commons.app.js" defer></script>
70. <script src="/nuxt-02/_nuxt/vendors.app.js" defer></script>
71. <script src="/nuxt-02/_nuxt/app.js" defer></script>
72. </body>
73. </html>
```

- ligne 47 : la valeur envoyée par le serveur ;
- ligne 60 : on constate que l'état du store **[Vuex]** a été embarqué dans la page. Cela va permettre au client qui va s'exécuter après réception de la page, de reconstituer un nouveau store **[Vuex]** avec 25 comme valeur initiale du compteur ;

Après réception et affichage de la page reçue du serveur, le client prend la main et exécute à son tour la page **[page1]** :

```
1.  ...
2.
3.  <script>
4.  /* eslint-disable no-console */
5.
6.  import Navigation from '@/components/navigation'
7.  import Layout from '@/components/layout'
8.
9.  export default {
10.   name: 'Page1',
11.   // composants utilisés
12.   components: {
13.     Layout,
14.     Navigation
15.   },
16.   data() {
17.     return {
18.       value: 0
19.     }
20.   },
21.   // cycle de vie
22.   beforeCreate() {
23.     // client et serveur
24.     console.log('[page1 beforeCreate]')
25.   },
26.   created() {
27.     // client et serveur
28.     console.log('[page1 created]')
29.     // serveur seulement
30.     if (process.server) {
31.       this.$store.commit('increment', 25)
32.     }
33.     // client et serveur
34.     this.value = this.$store.state.counter
35.     console.log('value=', this.value)
36.   },
37.   beforeMount() {
```

```
38.     // client seulement
39.     console.log('[page1 beforeMount]')
40.   },
41.   mounted() {
42.     // client seulement
43.     console.log('[page1 mounted]')
44.   }
45. }
46. </script>
```

- ligne 34 : la propriété **[value]** de la ligne 18 reçoit à son tour la valeur 25 du compteur ;

Le store de **[nuxt]** permet donc au serveur de transmettre des informations au client lors du chargement initial de la page, lorsque celle-ci est cherchée sur le serveur. On rappelle qu'une fois cette page obtenue, le serveur n'est plus sollicité et l'application fonctionne comme une application **[vue]** classique, en mode SAP.

4.5.3 La page [page2]

Dans la page **[page2]**, nous montrons une autre façon pour que :

- le serveur inclut des informations calculées dans la page ;
- le client ne modifie pas celles-ci ;

4.5.3.1 Le code de la page

Le code de la page **[page2]** évolue de la façon suivante :

```
1.  <!-- page2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 2 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page2',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   asyncData(context) {
26.     // qui exécute ce code ?
27.     console.log('asyncData, client=', process.client, 'serveur=', process.server)
28.     // seulement pour le serveur
29.     if (process.server) {
30.       // on retourne une promesse
31.       return new Promise(function(resolve, reject) {
32.         // on a normalement ici une fonction asynchrone
33.         // on la simule avec une attente d'une seconde
34.         setTimeout(() => {
35.           // ce résultat sera inclus dans les propriétés de [data]
36.           resolve({ value: 87 })
37.           // log
38.           console.log('asynData terminée')
39.         }, 1000)
40.       })
41.     }
42.   },
43.   // cycle de vie
44.   beforeCreate() {
45.     // client et serveur
46.     console.log('[page2 beforeCreate]')
47.   },
```

```
48.   created() {
49.     // client et serveur
50.     console.log('[page2 created]')
51.   },
52.   beforeMount() {
53.     // client seulement
54.     console.log('[page2 beforeMount]')
55.   },
56.   mounted() {
57.     // client seulement
58.     console.log('[page2 mounted]')
59.   }
60. }
61. </script>
```

- ligne 7 : la page affiche la valeur d'une propriété nommée **[value]** ;
- la propriété **[value]** n'existe pas comme élément d'un objet rendu par la fonction **[data]**. Ici cette fonction n'existe pas. La propriété **[value]** est créée dynamiquement par la ligne 36 ;
- ligne 25 : la fonction **[asyncData]** est une fonction **[nuxt]**. Comme son nom l'indique, c'est normalement une fonction asynchrone. Son rôle habituel est d'aller chercher des données externes. **[nuxt]** assure que la page n'est pas envoyée au navigateur client avant que la fonction **[asyncData]** n'ait rendu ses données asynchrones ;
- la fonction **[asyncData]** reçoit comme paramètre le contexte **[nuxt]**. Cet objet est très dense et donne accès à beaucoup d'informations sur l'application **[nuxt. Nous le découvrirons dans les sections à venir ;**
- **ligne 31 : on implémente la fonction [asyncData]** avec une **[Promise]** (cf document |Introduction au langage **ECMASCRIPT 6** par l'exemple|). Le constructeur de cette classe accepte comme paramètre une fonction asynchrone qui :
  - signale un succès en rendant des données avec la fonction **[resolve]**. L'objet rendu par cette fonction est automatiquement inclus dans les propriétés **[data]** de la page ;
  - signale un échec en rendant une erreur avec la fonction **[reject]** ;
- ligne 34 : on simule une fonction asynchrone avec la fonction **[setTimeout]**. Cette fonction rend l'objet **[{ value: 87 }]** (ligne 36) au bout d'une seconde (ligne 31) grâce à la fonction **[resolve]** qui signale un succès de la **[Promise]**. L'objet rendu par la fonction asynchrone est inclus automatiquement dans les propriétés **[data]** de la page. Et c'est donc cette propriété que la ligne 7 affiche ;
- ligne 27 : nous allons découvrir que la fonction **[asyncData]** est exécutée par le serveur mais pas par le client ;
- ligne 29 : l'initialisation de la propriété **[value]** est faite par le serveur ;

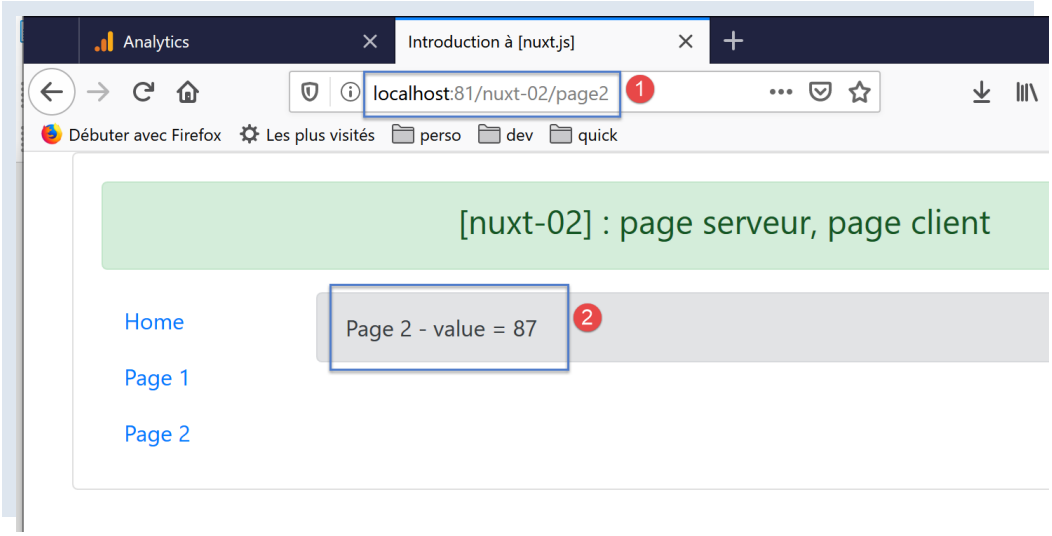
Note : l'objet **[this]** n'est pas connu dans la fonction **[asyncData]** car l'objet encapsulant le composant **[vue]** n'a pas encore été créé.

4.5.3.2 Exécution

On exécute le projet **[nuxt-02]** et on tape **[localhost:81/nuxt-02/page2]** à la main pour que le serveur soit sollicité. Comme au démarrage pour la page **[index]** :

- le serveur exécute la page **[page2.vue]** ;
- envoie la page générée au navigateur. Celle-ci est affichée ;
- les scripts client embarqués dans la page envoyée prennent la main et exécutent à nouveau la page **[page2.vue]** ;
- la page affichée est alors modifiée ;

Le résultat final est le suivant :



Cette fois-ci, la valeur affichée est bien celle fixée par le serveur et visuellement, on ne voit pas la page 'tressauter' à cause d'un changement par le client de la valeur affichée par le serveur. Cette fois-ci que s'est-il passé ?

Le serveur a exécuté la page **[page2]** suivante :

```
1.  <!-- page2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 2 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page2',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   asyncData(context) {
26.     // qui exécute ce code ?
27.     console.log('asyncData, client=', process.client, 'serveur=', process.server)
28.     // seulement pour le serveur
29.     if (process.server) {
30.       // on retourne une promesse
31.       return new Promise(function(resolve, reject) {
32.         // on a normalement ici une fonction asynchrone
33.         // on la simule avec une attente d'une seconde
34.         setTimeout(() => {
35.           // ce résultat sera inclus dans les propriétés de [data]
36.           resolve({ value: 87 })
37.           // log
38.           console.log('asynData terminée')
39.         }, 1000)
40.       })
41.     }
42.   },
43.   // cycle de vie
44.   beforeCreate() {
45.     // client et serveur
46.     console.log('[page2 beforeCreate]')
47.   },
48.   created() {
```

```
49.     // client et serveur
50.     console.log('[page2 created]')
51.   },
52.   beforeMount() {
53.     // client seulement
54.     console.log('[page2 beforeMount]')
55.   },
56.   mounted() {
57.     // client seulement
58.     console.log('[page2 mounted]')
59.   }
60. }
61. </script>
```

C'est la ligne 36 qui a fixé la valeur affichée par la ligne 7. C'est donc ce qu'a reçu le navigateur client. Très exactement il reçoit la page suivante :

```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata
    middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-02/">
10.   <link rel="preload" href="/nuxt-02/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-02/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-02/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-02/_nuxt/app.js" as="script">
14.   ...
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-
    success">
23.               <h4>[nuxt-02] : page serveur, page client</h4>
24.             </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
29.                     <li class="nav-item">
30.                       <a href="/nuxt-02/" target="_self" class="nav-link">
31.                         Home
32.                       </a>
33.                     </li>
34.                     <li class="nav-item">
35.                       <a href="/nuxt-02/page1" target="_self" class="nav-link">
36.                         Page 1
37.                       </a>
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-02/page2" target="_self" class="nav-link active nuxt-link-active">
41.                         Page 2
42.                       </a>
43.                     </li>
44.                   </ul>
45.                 </div>
46.                 <div class="col-10">
47.                   <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-secondary">
48.                     Page 2 - value = 87
49.                   </div>
50.                 </div>
51.               </div>
52.             </div>
53.           </div>
54.         </div>
55.       </div>
56.     </div>
57.   </div>
```

```
58.   <script>
59.   window.__NUXT__ = (function (a, b, c) {
60.       return {
61.         layout: "default", data: [{ value: 87 }], error: null, state: { counter: 0 }, serverRendered: true,
62.         logs: [
63.           { date: new Date(1574096608555), args: ["asyncData, client=", "false", "serveur=", "true"], type:
    a, level: b, tag: c },
64.           { date: new Date(1574096608575), args: ["[page2 beforeCreate]"], type: a, level: b, tag: c },
65.           { date: new Date(1574096608599), args: ["[page2 created]"], type: a, level: b, tag: c }
66.         ]
67.       }
68.     }("log", 2, ""));</script>
69.   <script src="/nuxt-02/_nuxt/runtime.js" defer></script>
70.   <script src="/nuxt-02/_nuxt/commons.app.js" defer></script>
71.   <script src="/nuxt-02/_nuxt/vendors.app.js" defer></script>
72.   <script src="/nuxt-02/_nuxt/app.js" defer></script>
73. </body>
74. </html>
```

- ligne 48 : on voit que la valeur dans la page reçue est 87 ;
- ligne 61 : dans la réponse du serveur, on voit deux objets : **[data]** et **[state]** :
  - **[state]** est l'état du store **[Vuex]**. Celui-ci a été instancié à partir du contenu du dossier **[store]** de l'application **[nuxt-02]** ;
  - **[data]** contient les propriétés créées par le serveur grâce à la fonction **[asyncData]**. On retrouve la propriété **[value : 87]** créée par le serveur. Les scripts du client vont intégrer cette propriété dans celles de la page **[page2]** ;

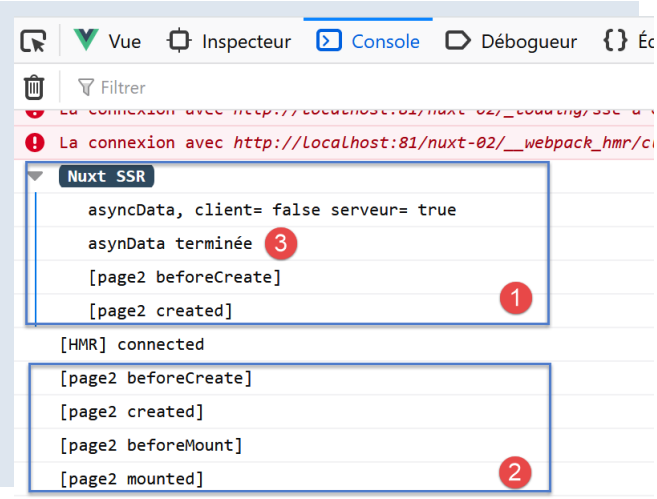
Revenons au code de la page **[page2]** :

```
1.  <!-- page2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 2 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page2',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   asyncData(context) {
26.     // qui exécute ce code ?
27.     console.log('asyncData, client=', process.client, 'serveur=', process.server)
28.     // seulement pour le serveur
29.     if (process.server) {
30.       // on retourne une promesse
31.       return new Promise(function(resolve, reject) {
32.         // on a normalement ici une fonction asynchrone
33.         // on la simule avec une attente d'une seconde
34.         setTimeout(() => {
35.           // ce résultat sera inclus dans les propriétés de [data]
36.           resolve({ value: 87 })
37.           // log
38.           console.log('asynData terminée')
39.         }, 1000)
40.       })
41.     }
42.   },
43.   // cycle de vie
44.   beforeCreate() {
45.     // client et serveur
46.     console.log('[page2 beforeCreate]')
47.   },
```

```
48.   created() {
49.     // client et serveur
50.     console.log('[page2 created]')
51.   },
52.   beforeMount() {
53.     // client seulement
54.     console.log('[page2 beforeMount]')
55.   },
56.   mounted() {
57.     // client seulement
58.     console.log('[page2 mounted]')
59.   }
60. }
61. </script>
```

- la ligne 7 utilise la propriété **[value]**. Or la page ne définit aucune propriété nommée **[value]**. Cependant les scripts du client ont créé automatiquement cette propriété grâce à l'objet **[data: [{ value: 87 }]]** reçue du serveur ;

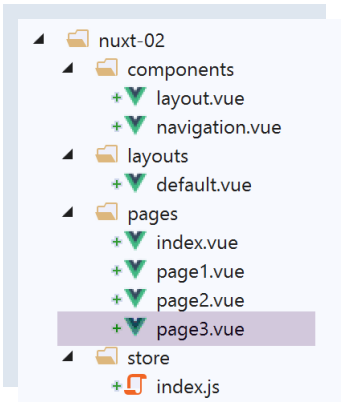
Les logs montrent par ailleurs que la fonction **[asyncData]** n'a pas été exécutée par le client :



La fonction **[asyncData]** a été exécutée par le serveur **[1]** mais pas par le client **[2]**. Par ailleurs, on peut noter que les fonctions du cycle de vie ne sont pas exécutées par le serveur avant la fin de la fonction **[asyncData]**. On peut augmenter la durée de l'attente au sein de la fonction **[asyncData]** pour le vérifier.

4.5.4 La page [page3]

Nous ajoutons une nouvelle page **[page3]** à notre application :



4.5.4.1 Le composant [navigation]

Le composant **[navigation]** est modifié pour permettre la navigation vers la nouvelle page :

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
```

```
    <b-nav vertical>
      <b-nav-item to="/" exact exact-active-class="active">
        Home
      </b-nav-item>
      <b-nav-item to="/page1" exact exact-active-class="active">
        Page 1
      </b-nav-item>
      <b-nav-item to="/page2" exact exact-active-class="active">
        Page 2
      </b-nav-item>
      <b-nav-item to="/page3" exact exact-active-class="active">
        Page 3
      </b-nav-item>
    </b-nav>
  </template>
```

4.5.4.2 Le code de [page3]

Le code de la page **[page3]** est le suivant :

```
  <!-- page3 -->
  <template>
    <Layout :left="true" :right="true">
      <!-- navigation -->
      <Navigation slot="left" />
      <!-- message -->
      <b-alert slot="right" show variant="secondary"> Page 3 - value = {{ value }} </b-alert>
    </Layout>
  </template>

  <script>
  /* eslint-disable no-console */
  /* eslint-disable nuxt/no-timing-in-fetch-data */

  import Navigation from '@/components/navigation'
  import Layout from '@/components/layout'

  export default {
    name: 'Page3',
    // composants utilisés
    components: {
      Layout,
      Navigation
    },
    data() {
      return {
        value: 0
      }
    },
    fetch(context) {
      // qui exécute ce code ?
      console.log('fetch, client=', process.client, 'serveur=', process.server)
      // seulement pour le serveur
      if (process.server) {
        // on retourne une promesse
        return new Promise(function(resolve, reject) {
          // on a normalement ici une fonction asynchrone
          // on la simule avec une attente d'une seconde
          setTimeout(() => {
            // succès
            resolve()
          }, 1000)
        }).then(() => {
          // on modifie le store
          context.store.commit('increment', 28)
          // log
          console.log('fetch commit terminé')
        })
      }
    },
    // cycle de vie
    beforeCreate() {
      // client et serveur
      console.log('[page3 beforeCreate]')
    },
    created() {
      // client et serveur
```

```
58.     this.value = this.$store.state.counter
59.     console.log('[page3 created], value=', this.value)
60.   },
61.   beforeMount() {
62.     // client seulement
63.     console.log('[page3 beforeMount]')
64.   },
65.   mounted() {
66.     // client seulement
67.     console.log('[page3 mounted]')
68.   }
69. }
70. </script>
```

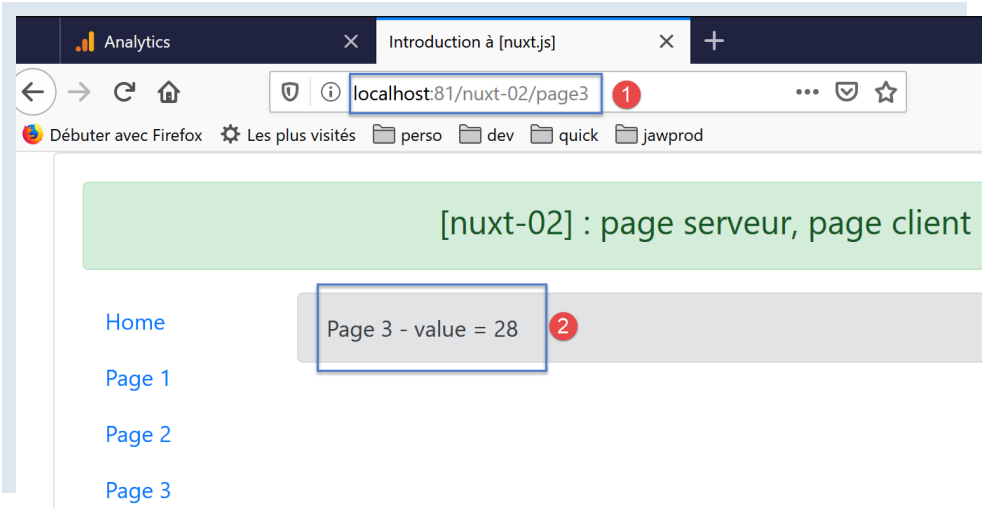
- ligne 30 : la fonction **[fetch]** a un comportement analogue à celui de la fonction **[asyncData]** :
  - elle est exécutée avant les fonctions du cycle de vie ;
  - l'objet **[this]** n'est pas connu dans cette fonction ;
  - son fonctionnement est asynchrone ;
  - le cycle de vie ne commence pas tant que la fonction asynchrone n'a pas rendu son résultat ;
  - le résultat est ici rendu par la méthode **[then]** de la **[Promise]**, ligne 43 ;
- fonction **[fetch]** reçoit le paramètre **[context]**. Celui-ci représente le contexte **[nuxt]** du moment ;
- ligne 30 : parmi ses nombreuses propriétés, l'objet **[context]** a une propriété **[store]** qui représente le store **[Vuex]** de l'application ;
- ligne 41 : artificiellement, on signale le succès de la **[Promise]** au bout d'une seconde (cf. document |Introduction au langage **ECMASCRIPT 6** par l'exemple|) ;
- ligne 45 : la méthode **[then]** est alors exécutée. On y incrémente le compteur du **[store]** ;

4.5.4.3 Exécution

On exécute le projet **[nuxt-02]** et on tape **[localhost:81/nuxt-02/page3]** à la main pour que le serveur soit sollicité. Comme au démarrage pour la page **[index]** :

- le serveur exécute la page **[page3.vue]** ;
- envoie la page générée au navigateur. Celle-ci est affichée ;
- les scripts client embarqués dans la page envoyée prennent la main et exécutent à nouveau la page **[page3.vue]** ;
- la page affichée est alors modifiée ;

Le résultat final est le suivant :



La valeur affichée est bien celle fixée par le serveur et visuellement, on ne voit pas la page 'tressauter' à cause d'un changement par le client de la valeur affichée par le serveur. Cette fois-ci que s'est-il passé ?

Le serveur a exécuté la page **[page3]** suivante :

```
1.  <!-- page3 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
```

```
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 3 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page3',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   data() {
26.     return {
27.       value: 0
28.     }
29.   },
30.   fetch(context) {
31.     // qui exécute ce code ?
32.     console.log('fetch, client=', process.client, 'serveur=', process.server)
33.     // seulement pour le serveur
34.     if (process.server) {
35.       // on retourne une promesse
36.       return new Promise(function(resolve, reject) {
37.         // on a normalement ici une fonction asynchrone
38.         // on la simule avec une attente d'une seconde
39.         setTimeout(() => {
40.           // succès
41.           resolve()
42.         }, 1000)
43.       }).then(() => {
44.         // on modifie le store
45.         context.store.commit('increment', 28)
46.         // log
47.         console.log('fetch commit terminé')
48.       })
49.     }
50.   },
51.   // cycle de vie
52.   beforeCreate() {
53.     // client et serveur
54.     console.log('[page3 beforeCreate]')
55.   },
56.   created() {
57.     // client et serveur
58.     this.value = this.$store.state.counter
59.     console.log('[page3 created], value=', this.value)
60.   },
61.   beforeMount() {
62.     // client seulement
63.     console.log('[page3 beforeMount]')
64.   },
65.   mounted() {
66.     // client seulement
67.     console.log('[page3 mounted]')
68.   }
69. }
70. </script>
```

- ligne 45 : la fonction asynchrone **[fetch]** est la première des fonctions ci-dessus à s'exécuter. Elle reçoit en paramètre, un objet appelé **[context]** qui est le contexte **[nuxt]** du moment. Parmi les très nombreuses propriétés de cet objet, la propriété **[context.store]** représente le store **[Vuex]** ;
- ligne 45 : dans la fonction asynchrone **[fetch]**, le serveur fixe le compteur du store à 28 ;
- ligne 56 : lorsque la fonction **[created]** s'exécute, **[nuxt]** garantit que la fonction asynchrone **[fetch]** a terminé son travail ;
- ligne 58 : la valeur du compteur du store est affectée à la propriété **[value]** de la ligne 27 ;
- ligne 7 : affichage de la valeur de **[value]**, donc du compteur du store ;

Le navigateur client reçoit la page suivante :

```
<!doctype html>
<html data-n-head-ssr>
<head>
  <title>Introduction à [nuxt.js]</title>
  <meta data-n-head="ssr" charset="utf-8">
  <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
  <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata middleware plugins store">
  <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
  <base href="/nuxt-02/">
  <link rel="preload" href="/nuxt-02/_nuxt/runtime.js" as="script">
  <link rel="preload" href="/nuxt-02/_nuxt/commons.app.js" as="script">
  <link rel="preload" href="/nuxt-02/_nuxt/vendors.app.js" as="script">
  <link rel="preload" href="/nuxt-02/_nuxt/app.js" as="script">
  ...
</head>
<body>
  <div data-server-rendered="true" id="__nuxt">
    <div id="__layout">
      <div class="container">
        <div class="card">
          <div class="card-body">
            <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-success">
              <h4>[nuxt-02] : page serveur, page client</h4>
            </div>
            <div>
              <div class="row">
                <div class="col-2">
                  <ul class="nav flex-column">
                    <li class="nav-item">
                      <a href="/nuxt-02/" target="_self" class="nav-link">
                        Home
                      </a>
                    </li>
                    <li class="nav-item">
                      <a href="/nuxt-02/page1" target="_self" class="nav-link">
                        Page 1
                      </a>
                    </li>
                    <li class="nav-item">
                      <a href="/nuxt-02/page2" target="_self" class="nav-link">
                        Page 2
                      </a>
                    </li>
                    <li class="nav-item">
                      <a href="/nuxt-02/page3" target="_self" class="nav-link active nuxt-link-active">
                        Page 3
                      </a>
                    </li>
                  </ul>
                </div> <div class="col-10">
                  <div role="alert" aria-live="polite" aria-atomic="true" class="alert alert-secondary">
                    Page 3 - value = 28
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
  <script>
  window.__NUXT__ = (function (a, b, c) {
      return {
        layout: "default", data: [{}], error: null, state: { counter: 28 }, serverRendered: true,
        logs: [
          { date: new Date(1574169916025), args: ["fetch, client=", "false", "serveur=", "true"], type: a, level: b, tag: c },
          { date: new Date(1574169917038), args: ["fetch commit terminé"], type: a, level: b, tag: c },
          { date: new Date(1574169917137), args: ["[page3 beforeCreate]"], type: a, level: b, tag: c },
          { date: new Date(1574169917167), args: ["[page3 created], value=", "28"], type: a, level: b, tag: c }
        ]
```

```
72.       }
73.     }("log", 2, ""));</script>
74.   <script src="/nuxt-02/_nuxt/runtime.js" defer></script>
75.   <script src="/nuxt-02/_nuxt/commons.app.js" defer></script>
76.   <script src="/nuxt-02/_nuxt/vendors.app.js" defer></script>
77.   <script src="/nuxt-02/_nuxt/app.js" defer></script>
78. </body>
79. </html>
```

- ligne 52 : on voit que la valeur dans la page reçue est 28 ;
- ligne 65 : dans la réponse du serveur, on voit que le serveur a envoyé au client l'état **[state]** du store **[Vuex]**. Grâce à cette information, les scripts client vont pouvoir reconstituer un store **[Vuex]** ;

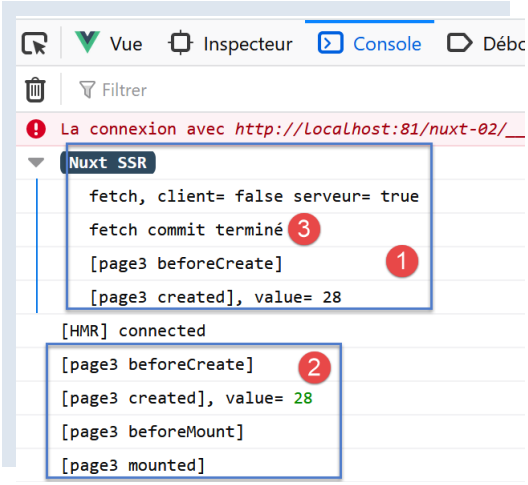
Les scripts client vont à leur tour exécuter le code de la page **[page3]** :

```
1.  <!-- page3 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary"> Page 3 - value = {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page3',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   data() {
26.     return {
27.       value: 0
28.     }
29.   },
30.   fetch(context) {
31.     // qui exécute ce code ?
32.     console.log('fetch, client=', process.client, 'serveur=', process.server)
33.     // seulement pour le serveur
34.     if (process.server) {
35.       // on retourne une promesse
36.       return new Promise(function(resolve, reject) {
37.         // on a normalement ici une fonction asynchrone
38.         // on la simule avec une attente d'une seconde
39.         setTimeout(() => {
40.           // succès
41.           resolve()
42.         }, 1000)
43.       }).then(() => {
44.         // on modifie le store
45.         context.store.commit('increment', 28)
46.         // log
47.         console.log('fetch commit terminé')
48.       })
49.     }
50.   },
51.   // cycle de vie
52.   beforeCreate() {
53.     // client et serveur
54.     console.log('[page3 beforeCreate]')
55.   },
56.   created() {
57.     // client et serveur
58.     this.value = this.$store.state.counter
59.     console.log('[page3 created], value=', this.value)
60.   },
61.   beforeMount() {
```

```
62.     // client seulement
63.     console.log('[page3 beforeMount]')
64.   },
65.   mounted() {
66.     // client seulement
67.     console.log('[page3 mounted]')
68.   }
69. }
70. </script>
```

- line 58 : la fonction **[created]** exécutée par le client affecte la valeur du compteur à la propriété **[value]** de la ligne 27 ;
- la ligne 7 affiche cette valeur. Puisque c'est la même que celle envoyée par le serveur, on ne voit pas la page 'tressauter' à cause d'une modification ;

Les logs montrent par ailleurs que la fonction **[fetch]** n'a pas été exécutée par le client :

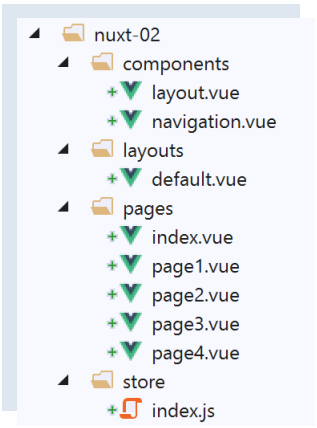


La fonction **[fetch]** a été exécutée par le serveur **[1]** mais pas par le client **[2]**. Par ailleurs, on peut noter que les fonctions du cycle de vie ne sont pas exécutées par le serveur avant la fin de la fonction **[fetch]** **[3]**. On peut augmenter la durée de l'attente au sein de la fonction **[fetch]** pour le vérifier.

Les pages **[page1]** et **[page3]** ont montré deux méthodes utilisant le store **[Vuex]** pour transmettre une information du serveur au client. On peut se demander si elles sont équivalentes. Nous construisons une page **[page4]** pour le vérifier.

4.5.5 La page [page4]

Nous ajoutons une nouvelle page **[page4]** à notre application :



4.5.5.1 Le composant [navigation]

Le composant **[navigation]** est modifié pour permettre la navigation vers la nouvelle page :

```
1.  <template>
2.    <!-- menu Bootstrap à cinq options -->
```

```
    <b-nav vertical>
      <b-nav-item to="/" exact exact-active-class="active">
        Home
      </b-nav-item>
      <b-nav-item to="/page1" exact exact-active-class="active">
        Page 1
      </b-nav-item>
      <b-nav-item to="/page2" exact exact-active-class="active">
        Page 2
      </b-nav-item>
      <b-nav-item to="/page3" exact exact-active-class="active">
        Page 3
      </b-nav-item>
      <b-nav-item to="/page4" exact exact-active-class="active">
        Page 4
      </b-nav-item>
    </b-nav>
  </template>
```

4.5.5.2 Le code de [page4]

Le code de la page **[page4]** est le suivant :

```
  <!-- page4 -->
  <template>
    <Layout :left="true" :right="true">
      <!-- navigation -->
      <Navigation slot="left" />
      <!-- message -->
      <b-alert slot="right" show variant="secondary"> Page 4 - value = {{ value }} </b-alert>
    </Layout>
  </template>

  <script>
  /* eslint-disable no-console */

  import Navigation from '@/components/navigation'
  import Layout from '@/components/layout'

  export default {
    name: 'Page4',
    // composants utilisés
    components: {
      Layout,
      Navigation
    },
    data() {
      return {
        value: 0
      }
    },
    // cycle de vie
    async beforeCreate() {
      // client et serveur
      console.log('[page4 beforeCreate]')
      // seulement pour le serveur
      // if (process.server) {
      // on exécute la fonction asynchrone
      const valeur = await new Promise(function(resolve, reject) {
        // on a normalement ici une fonction asynchrone
        // on la simule avec une attente de 10 secondes
        setTimeout(() => {
          // succès - on rend la valeur du compteur
          resolve(52)
        }, 10000)
      })
      // on modifie le store
      this.$store.commit('increment', valeur)
      // log
      console.log('[page4 beforeCreate], fonction asynchrone terminée, compteur=', this.$store.state.counter)
      // }
    },
    created() {
      // client et serveur
      this.value = this.$store.state.counter
      console.log('[page4 created], value=', this.value)
```

```
54.   },
55.   beforeMount() {
56.     // client seulement
57.     console.log('[page4 beforeMount]')
58.   },
59.   mounted() {
60.     // client seulement
61.     console.log('[page4 mounted]')
62.   }
63. }
64. </script>
```

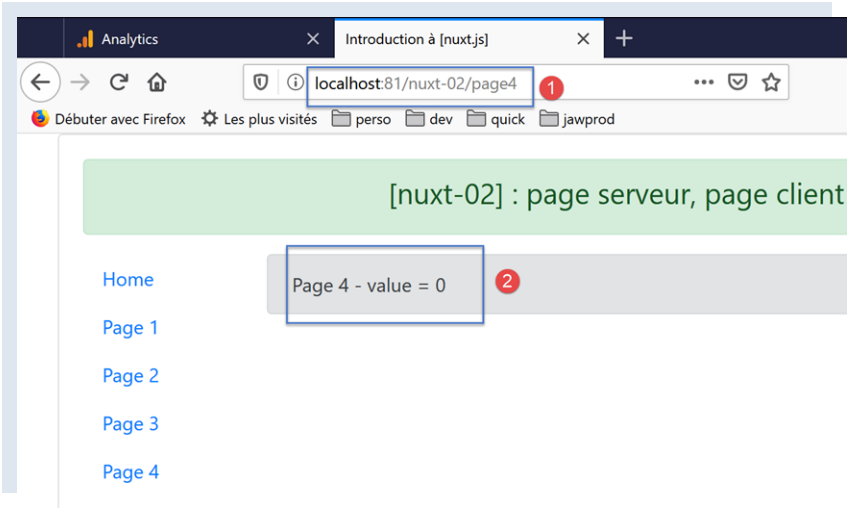
- ligne 30 : ce qui était fait auparavant dans la fonction **[fetch]** est désormais fait dans la méthode **[beforeCreate]**. On utilise le couple async (ligne 30) / await (ligne 36) pour attendre la fin de la fonction asynchrone ;
- ligne 36 : on récupère le résultat de la fonction asynchrone rendu ligne 41 après 10 secondes (ligne 42) ;
- lignes 50-54 : dans la méthode **[created]** exécutée aussi bien côté serveur que côté client, le compteur est affecté à la propriété **[value]** de la page ;

4.5.5.3 Exécution

On exécute le projet **[nuxt-02]** et on tape **[localhost:81/nuxt-02/page4]** à la main pour que le serveur soit sollicité. Comme au démarrage pour la page **[index]** :

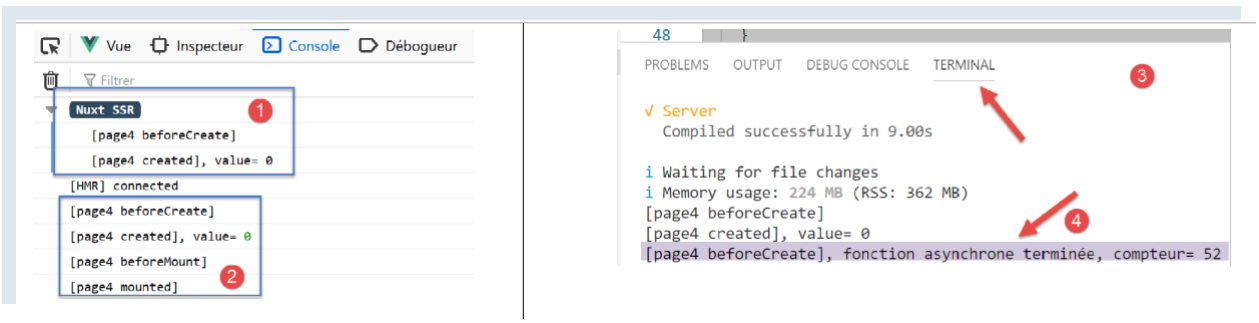
- le serveur exécute la page **[page4.vue]** ;
- envoie la page générée au navigateur. Celle-ci est affichée ;
- les scripts client embarqués dans la page envoyée prennent la main et exécutent à nouveau la page **[page4.vue]** ;
- la page affichée est alors modifiée ;

Le résultat final est le suivant :



Contrairement à ce qui était attendu, la valeur affichée en **[2]** n'est pas 52. Que s'est-il passé ?

Les logs sont les suivants :



On peut remarquer qu'en **[1]** le log de fin de l'action asynchrone n'a pas été affiché. La fonction **[created]** qui affiche la valeur du compteur, affiche 0. Tout cela laisse penser que **[nuxt]** n'a pas attendu la fin de l'action asynchrone.

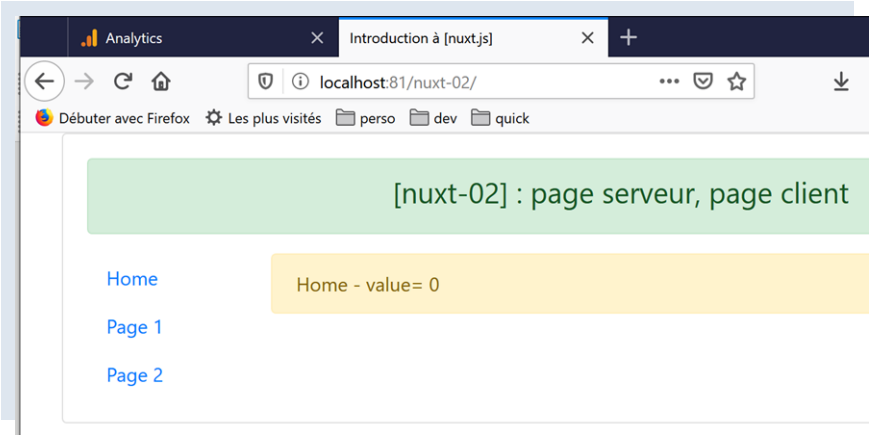
Si on retourne dans le terminal de VSCode qui a servi au lancement de l'application, on trouve les logs **[3-4]**. On voit que la fonction asynchrone a bien été exécutée côté serveur.

Au final, la fonction **[beforeCreate]** a bien été exécutée totalement côté serveur, mais **[nuxt]** n'a pas attendu la fin de son exécution pour envoyer la page au navigateur client alors qu'il attend bien la fin de la fonction **[fetch]**. C'est donc cette méthode qu'il faut utiliser si on veut que le serveur initialise un store **[Vuex]**.

4.5.6 Navigation dans l'application [vue]

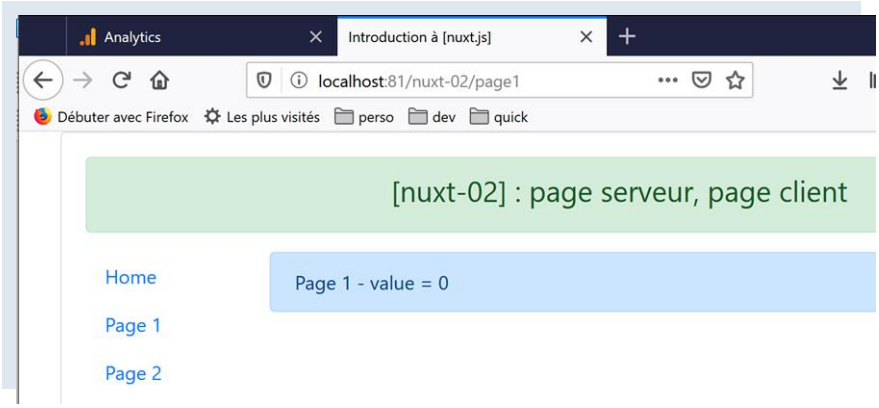
Nous avons montré ce qui se passait lorsque chacune des pages **[index, page1, page2, page3, page4]** était chargée initialement par le serveur. Dans la pratique, ce n'est pas ce qui se passe : dans un fonctionnement normal, seule la page **[index]** est cherchée sur le serveur. Voyons les trois pages dans ce cas là :

page [index]



Nous avons déjà expliqué ce résultat au paragraphe lien.

Maintenant cliquons sur le lien **[Page 1]** :



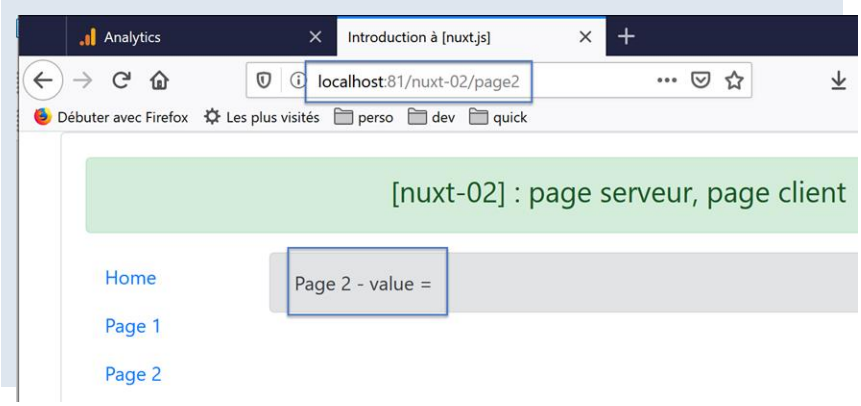
La valeur affichée est 0. C'était 25 lorsque la page était d'abord demandée au serveur en tapant son URL à la main. L'explication est simple. Le code exécuté est le suivant :

```
1.    created() {
2.      // client et serveur
3.      console.log('[page1 created]')
4.      // serveur seulement
5.      if (process.server) {
6.        this.$store.commit('increment', 25)
7.      }
8.      // client et serveur
```

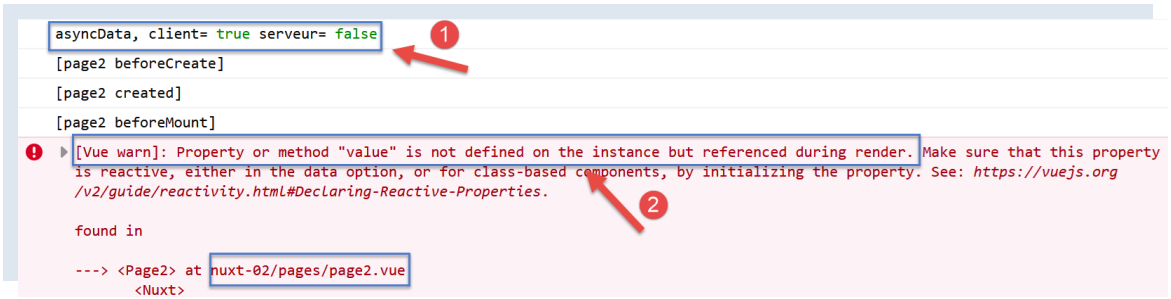
```
9.      this.value = this.$store.state.counter
10.     console.log('value=', this.value)
11. },
```

C'est la ligne 6 qui mettait le compteur à 25. Comme la page n'a pas été demandée au serveur, les lignes 5-7 n'ont pas été exécutées et le compteur du store **[Vuex]** est resté à 0.

Maintenant, cliquons sur le lien **[Page 2]** :



Cette fois-ci, aucune valeur n'est affichée et on a de plus un avertissement dans les logs de la console :



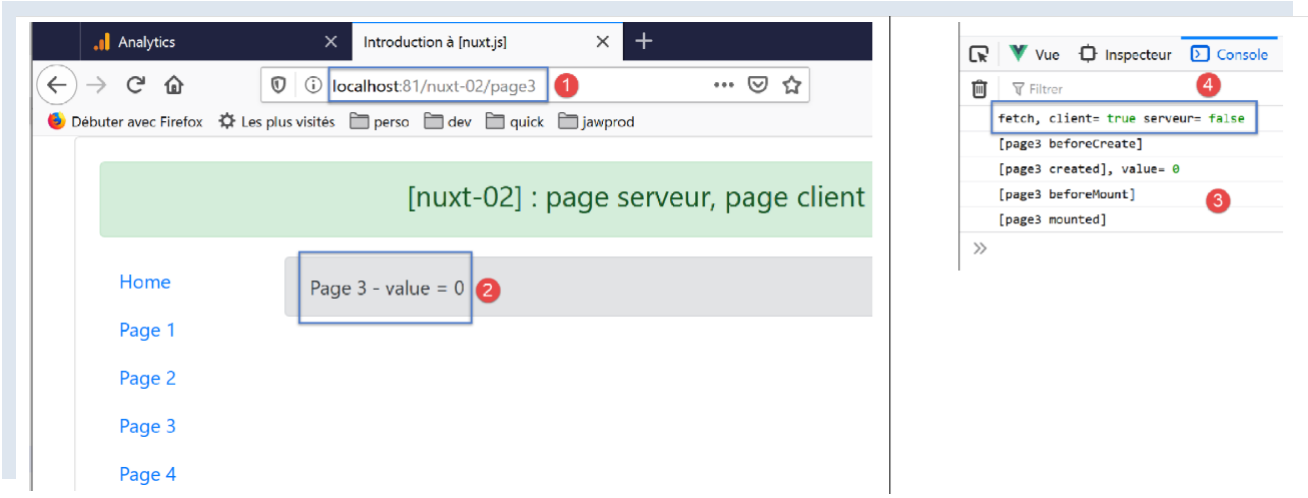
- en **[1]**, on découvre que la fonction **[asyncData]** a été exécutée par le client. C'est toujours ainsi :
- elle est exécutée par le serveur si la page est demandée au serveur. dans ce cas, elle n'est pas exécutée par le client ;
- puis à chaque fois que la page est la **cible** de la route courante du client ;
- en **[2]** : **[nuxt]** émet un avertissement parce que dans le template de la page, il y a une expression réactive `{{ value }}` alors que la page n'a pas de propriété **[value]** ;

Rappelons le code exécuté par le client :

```
1.  asyncData() {
2.      // qui exécute ce code ?
3.      console.log('asyncData, client=', process.client, 'serveur=', process.server)
4.      // seulement pour le serveur
5.      if (process.server) {
6.        // on retourne une promesse
7.        return new Promise(function(resolve, reject) {
8.          // on a normalement ici une fonction asynchrone - pas ici donc
9.          // ce résultat sera inclus dans les propriétés de [data]
10.         resolve({ value: 87 })
11.       })
12.     }
13. },
```

- la ligne 3 a permis de voir que la fonction **[asyncData]** a été exécutée par le client avant même les fonctions du cycle de vie ;
- la ligne 5 a empêché l'exécution du reste du code qui créait la propriété **[value]** car ce code est exécuté par le client ;

Maintenant passons à la page **[page3]** :



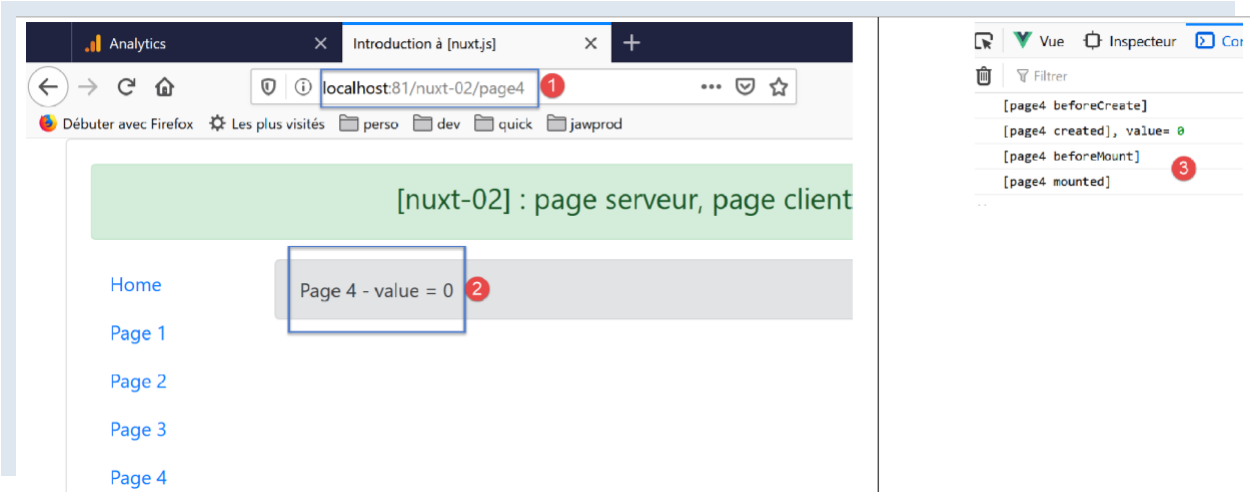
- en **[2]**, on avait 28 lorsque la page était fournie par le serveur. Ici ce n'est pas le cas ;
- en **[4]**, on voit que la fonction **[fetch]** a été exécutée côté client ;

Examinons le code exécuté par le client :

```
1.  ...
2.  fetch(context) {
3.      // qui exécute ce code ?
4.      console.log('fetch, client=', process.client, 'serveur=', process.server)
5.      // seulement pour le serveur
6.      if (process.server) {
7.        // on retourne une promesse
8.        return new Promise(function(resolve, reject) {
9.          // on a normalement ici une fonction asynchrone
10.         // on la simule avec une attente d'une seconde
11.         setTimeout(() => {
12.           // succès
13.           resolve()
14.         }, 1000)
15.       }).then(() => {
16.         // on modifie le store
17.         context.store.commit('increment', 28)
18.         // log
19.         console.log('fetch commit terminé')
20.       })
21.     }
22. },
23. ...
```

- le client exécute la méthode **[fetch]**, ligne 2 ;
- les lignes 6-21 ne sont pas exécutées car la condition **[process.server]** est fausse. Donc la ligne 17 qui met le compteur à 28 n'est pas exécuté. Il reste à zéro. C'est pourquoi le client affiche 0 au lieu de 28 ;

Passons maintenant à la page **[page4]**. On a le résultat suivant :



- en **[2]**, la valeur du compteur ;
- en **[3]**, les logs du client ;

Le code exécuté par le client est le suivant :

```
1.  ...
2.  data() {
3.      return {
4.        value: 0
5.      }
6.    },
7.    // cycle de vie
8.    async beforeCreate() {
9.      // client et serveur
10.     console.log('[page4 beforeCreate]')
11.     // seulement pour le serveur
12.     if (process.server) {
13.       // on exécute la fonction asynchrone
14.       const valeur = await new Promise(function(resolve, reject) {
15.         // on a normalement ici une fonction asynchrone
16.         // on la simule avec une attente de 10 secondes
17.         setTimeout(() => {
18.           // succès - on rend la valeur du compteur
19.           resolve(52)
20.         }, 10000)
21.       })
22.       // on modifie le store
23.       this.$store.commit('increment', valeur)
24.       // log
25.       console.log('[page4 beforeCreate], fonction asynchrone terminée, compteur=',
    this.$store.state.counter)
26.     }
27.   },
28.   created() {
29.     // client et serveur
30.     this.value = this.$store.state.counter
31.     console.log('[page4 created], value=', this.value)
32.   },
33. ...
```

- les lignes 12-26 ne sont pas exécutées par le client, car la condition **[process.server]** de la ligne 12 est fausse. Donc le compteur du store **[Vuex]** vaut 0 (sa valeur initiale dans le store) et c'est cette valeur qu'affiche la page ;

On peut se demander ce qui se passe lorsqu'on met en commentaires le **[if]** des lignes 12 et 26. Voici la réponse :

- le client exécute cette fois les lignes 14-25 mais **[nuxt]** n'attend pas la fin de la fonction asynchrone (comme pour le serveur) et laisse donc le compteur à 0 ;
- au bout de 10 s, la fonction asynchrone se termine et le compteur est mis à 52 ligne 23 ;
- lorsqu'on navigue de nouveau vers la page **[page4]**, c'est alors la valeur 52 qui est affichée ;

4.5.7 Résumé

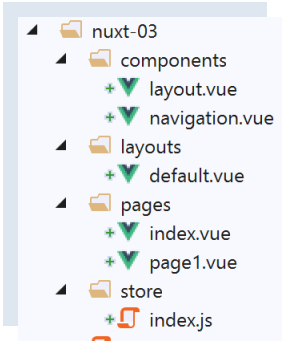
De nos différents essais, on retiendra les points suivants :

- si la page **[index]** doit contenir des données externes, celles-ci peuvent être cherchées par le serveur avec la fonction **[asyncData]** ;
- si le serveur doit initialiser un store **[Vuex]** avec des données externes au chargement de la page **[index]**, il le fera dans la fonction **[fetch]** ;
- la page générée par le serveur et la page générée par le client doivent être **identiques** si on veut éviter l'effet de 'tressautement' provoqué par le fait que la page du client vient visuellement remplacer la page envoyée par le serveur et initialement affichée ;

Nous allons découvrir d'autres aspects de **[nuxt]** au travers d'un nouvel exemple.

4.6 Exemple [nuxt-03] : nuxtServerInit

Le projet **[nuxt-03]** vise à présenter une fonction du store **[Vuex]** appelée **[nuxtServerInit]**. Elle permet au serveur d'initialiser le store **[Vuex]** comme le fait la fonction **[fetch]**. Mais contrairement à la fonction **[fetch]**, la fonction **[nuxtServerInit]** n'est jamais exécutée par le client.



Le projet **[nuxt-03]** est initialement obtenu par recopie du projet **[nuxt-01]** duquel on supprime la page **[page2]** du dossier **[pages]** et du composant **[navigation]**. Le dossier **[store]** est obtenu par recopie du dossier **[nuxt-02/store]**.

4.6.1 Le store [Vuex]

Le store **[Vuex]** sera implémenté par le fichier **[store/index.js]** suivant :

```
1.  /* eslint-disable no-console */
2.  export const state = () => ({
3.    // compteut
4.    counter: 0
5.  })
6.
7.  export const mutations = {
8.    // incrémentation du compteur d'une valeur [inc]
9.    increment(state, inc) {
10.     state.counter += inc
11.   }
12. }
13.
14. export const actions = {
15.   async nuxtServerInit(store, context) {
16.     // qui exécute ce code ?
17.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server)
18.     // on attend la fin d'une promesse
19.     await new Promise(function(resolve, reject) {
20.       // on a normalement ici une fonction asynchrone
21.       // on la simule avec une attente d'une seconde
22.       setTimeout(() => {
23.         // succès
24.         resolve()
25.       }, 1000)
26.     })
27.     // on modifie le store
28.     store.commit('increment', 34)
29.     // log
30.     console.log('nuxtServerInit commit terminé')
31.   }
```

```
32. }
```

- lignes 1-12 : sont analogues à ce qu'elles étaient dans le projet **[nuxt-02]** ;
- lignes 14-32 : on exporte un objet **[actions]**. C'est un terme réservé du store de **[Vuex]** ;
- ligne 15 : on définit la fonction **[nuxtServerInit]**. Celle-ci sera exécutée au démarrage de l'application par le serveur. Son rôle usuel est d'initialiser un store **[Vuex]** à l'aide de données externes obtenues avec une fonction asynchrone. **[nuxt]** attend que celle-ci rende ses résultats avant d'entamer le cycle de vie de la page demandée. La fonction reçoit deux paramètres :
- le store **[Vuex]** à initialiser ;
- le contexte **[nuxt]** du moment ;
- lignes 19-26 : on attend la fin de l'action asynchrone, ici une attente artificielle d'une seconde (ligne 15) ;
- ligne 28 : on donne au compteur la valeur 34 ;
- lignes 17 et 30 : des logs pour suivre le déroulement de l'exécution de la fonction **[nuxtServerInit]** ;

4.6.2 La page [index]

La page **[index]** sera la suivante :

```
1.  <!-- page [index] -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning"> Home - value= {{ value }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-undef */
13. /* eslint-disable no-console */
14. /* eslint-disable nuxt/no-env-in-hooks */
15.
16. import Layout from '@/components/layout'
17. import Navigation from '@/components/navigation'
18. export default {
19.   name: 'Home',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   data() {
26.     return {
27.       value: 0
28.     }
29.   },
30.   // cycle de vie
31.   beforeCreate() {
32.     // client et serveur
33.     console.log('[home beforeCreate]')
34.   },
35.   created() {
36.     // client et serveur
37.     this.value = this.$store.state.counter
38.     console.log('[home created], value=', this.value)
39.   },
40.   beforeMount() {
41.     // client seulement
42.     console.log('[home beforeMount]')
43.   },
44.   mounted() {
45.     // client seulement
46.     console.log('[home mounted]')
47.   }
48. }
49. </script>
```

- ligne 37 : la valeur du compteur initialisé par la fonction **[nuxtServerInit]** est affectée à la propriété **[value]** de la ligne 27. Cette valeur est affichée par la ligne 7 ;
- la ligne 37 sera exécutée aussi bien par le serveur que par le client. Dans les deux cas, la propriété **[value]** recevra la même valeur ce qui assure l'identité de la page générée par le serveur avec celle générée par le client ;

4.6.3 La page [page1]

La page **[page1]** est obtenue par recopie de la page **[index]**. On modifie ensuite son texte pour remplacer **[home]** par **[page1]** :

```
<!-- page [page1]] -->
<template>
  <Layout :left="true" :right="true">
    <!-- navigation -->
    <Navigation slot="left" />
    <!-- message-->
    <b-alert slot="right" show variant="warning"> Page1 - value= {{ value }} </b-alert>
  </Layout>
</template>

<script>
/* eslint-disable no-undef */
/* eslint-disable no-console */
/* eslint-disable nuxt/no-env-in-hooks */

import Layout from '@/components/layout'
import Navigation from '@/components/navigation'

export default {
  name: 'Page1',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  data() {
    return {
      value: 0
    }
  },
  // cycle de vie
  beforeCreate() {
    // client et serveur
    console.log('[page1 beforeCreate]')
  },
  created() {
    // client et serveur
    this.value = this.$store.state.counter
    console.log('[page1 created], value=', this.value)
  },
  beforeMount() {
    // client seulement
    console.log('[page1 beforeMount]')
  },
  mounted() {
    // client seulement
    console.log('[page1 mounted]')
  }
}
</script>
```

Cette page n'est là que pour rendre possible la navigation entre deux pages.

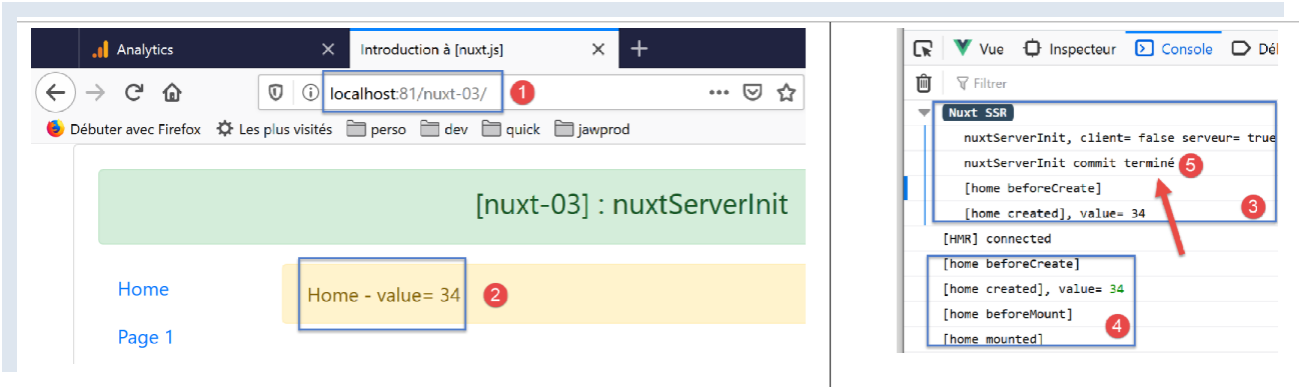
4.6.4 Exécution

Le fichier **[nuxt.config.js]** est modifié de la façon suivante :

```
  // répertoire du code source
  srcDir: 'nuxt-04',
  // routeur
  router: {
    // racine des URL de l'application
    base: '/nuxt-04/'
  },
  // serveur
  server: {
    // port de service, 3000 par défaut
    port: 81,
    // adresses réseau écoutées, par défaut localhost : 127.0.0.1
    // 0.0.0.0 = toutes les adresses réseau de la machine
    host: 'localhost'
```

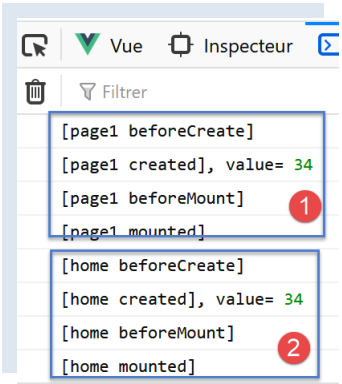
```
15.    }
```

La page affichée à l'exécution est alors la suivante :



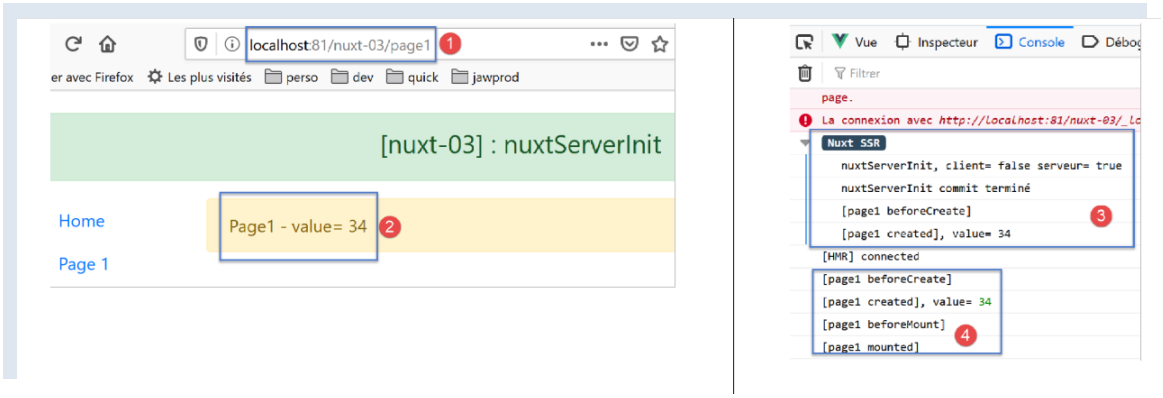
- en **[5]**, on voit que la fonction **[nuxtServerInit]** a été exécutée par le serveur avant le cycle de vie de la page **[index]**. **[nuxt]** a attendu que la fonction asynchrone ait terminé son travail avant de passer au cycle de vie ;
- en **[4]**, on voit que le client n'a pas exécuté la fonction **[nuxtServerInit]** ;

Maintenant naviguons deux fois : index --> page1 --> index. Les logs sont alors les suivants :



- en **[1-2]**, on voit que la fonction **[nuxtServerInit]** n'est pas exécutée par le client ;

Maintenant tapons l'URL de la page **[page1]** à la main pour forcer un appel au serveur :



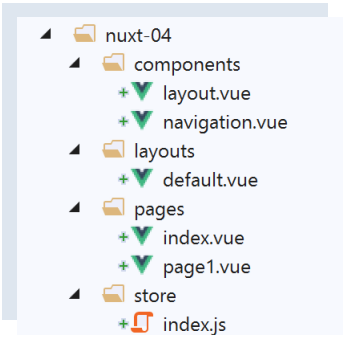
- en **[3-4]**, on retrouve le même mécanisme que celui qui avait précédé le chargement de la page **[index]** au démarrage. On rappelle ici ce qui a déjà été dit : lorsqu'on force l'appel d'une page au serveur, tout se passe comme si l'application redémarrait avec une page d'accueil qui serait la page demandée ;

4.7 Exemple [nuxt-04] : maintien d'une session client / serveur

Le projet **[nuxt-04]** aborde le problème du maintien d'une session client / serveur. On reprend le projet **[nuxt-03]** avec les modifications suivantes :

- la page **[index]** aura un bouton qui permettra d'incrémenter le compteur du store **[Vuex]** ;
- la page **[page1]** reste inchangée ;
- on voudrait que lorsqu'une page est demandée à la main au serveur, celui-ci renvoie la page demandée avec un store **[Vuex]** qui aurait pour valeur de compteur, la dernière valeur que celui-ci avait côté client ;

Le projet **[nuxt-04]** est initialement créé par recopie du projet **[nuxt-03]** :



Seule la page **[index]** change :

```
<!-- page [index] -->
<template>
  <Layout :left="true" :right="true">
    <!-- navigation -->
    <Navigation slot="left" />
    <!-- message-->
    <template slot="right">
      <b-alert show variant="warning"> Home - value= {{ value }} </b-alert>
      <!-- bouton -->
      <b-button @click="incrementCounter" class="ml-3" variant="primary">Incrémenter</b-button>
    </template>
  </Layout>
</template>

<script>
/* eslint-disable no-undef */
/* eslint-disable no-console */
/* eslint-disable nuxt/no-env-in-hooks */

import Layout from '@/components/layout'
import Navigation from '@/components/navigation'
export default {
  name: 'Home',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  data() {
    return {
      value: 0
    }
  },
  // cycle de vie
  beforeCreate() {
    // client et serveur
    console.log('[home beforeCreate]')
  },
  created() {
    // client et serveur
    this.value = this.$store.state.counter
    console.log('[home created], value=', this.value)
  },
  beforeMount() {
```

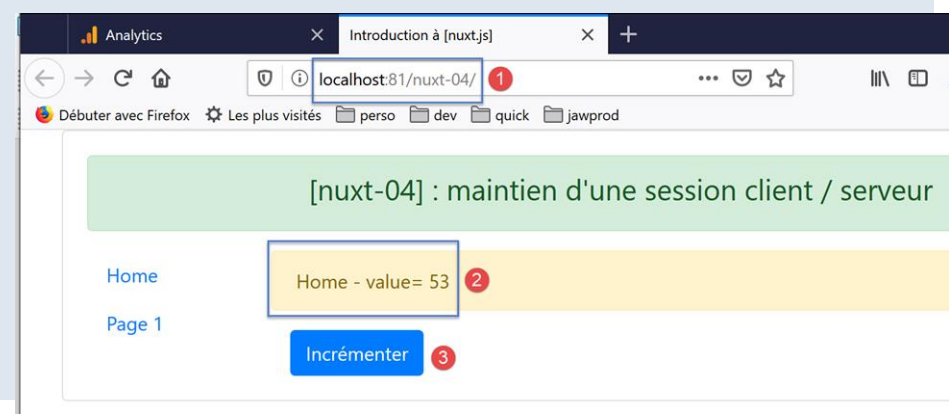
```
45.     // client seulement
46.     console.log('[home beforeMount]')
47.   },
48.   mounted() {
49.     // client seulement
50.     console.log('[home mounted]')
51.   },
52.   // gestion des évts
53.   methods: {
54.     incrementCounter() {
55.       console.log('incrementCounter')
56.       // incrément du compteur de 1
57.       this.$store.commit('increment', 1)
58.       // chgt de la valeur affichée
59.       this.value = this.$store.state.counter
60.     }
61.   }
62. }
63. </script>
```

- ligne 10 : on a ajouté un bouton pour incrémenter le compteur du store **[Vuex]** ;
- ligne 54 : la méthode qui gère le **[clic]** sur le bouton **[Incrémenter]** ;
- ligne 57 : le compteur du store est incrémenté de 1 ;
- ligne 59 : la valeur du compteur est affectée à la propriété **[value]** afin qu'elle soit affichée par la ligne 8 ;

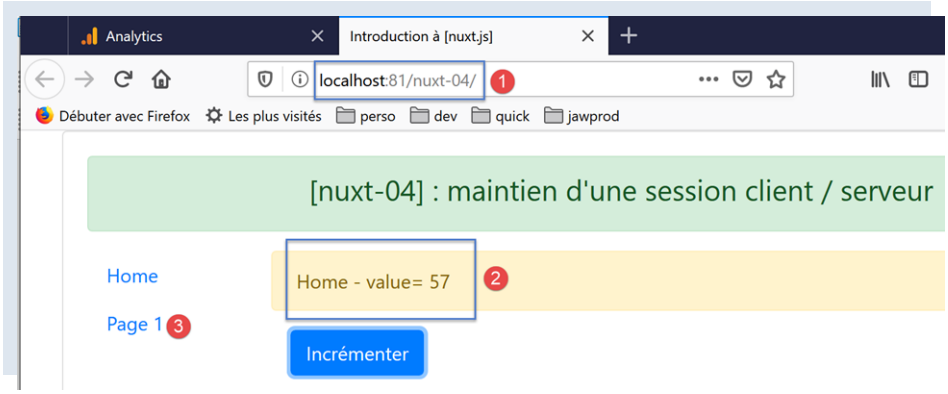
On exécute le projet **[nuxt-04]** avec le fichier **[nuxt.config.js]** suivant :

```
1.  // répertoire du code source
2.    srcDir: 'nuxt-04',
3.    // routeur
4.    router: {
5.      // racine des URL de l'application
6.      base: '/nuxt-04/'
7.    },
8.    // serveur
9.    server: {
10.     // port de service, 3000 par défaut
11.     port: 81,
12.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
13.     // 0.0.0.0 = toutes les adresses réseau de la machine
14.     host: 'localhost'
15. }
```

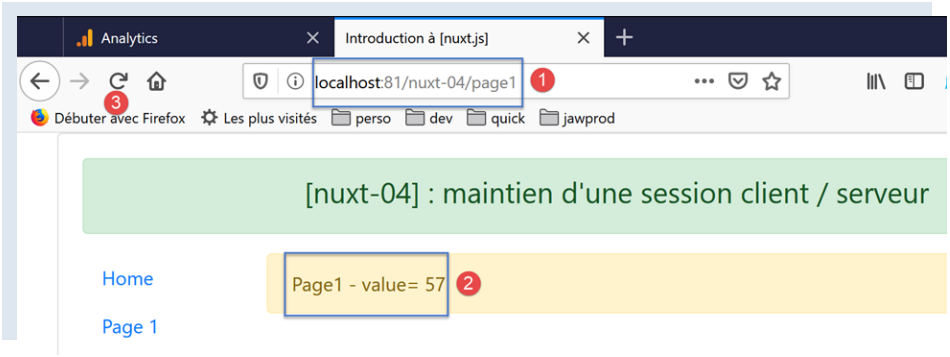
A l'exécution la première page affichée est la suivante :



En utilisant plusieurs fois le bouton **[3]**, on a la nouvelle page suivante :



Si on utilise le lien **[3]**, on a la page suivante :



- en **[2]**, la page **[page1] [1]** affiche bien la valeur du compteur ;

Maintenant, rafraîchissons la page avec **[3]**. La nouvelle page est la suivante :

- en **[2]**, on a perdu la valeur courante du compteur. On est revenu à sa valeur initiale ;

Ce résultat est tout a fait compréhensible si on regarde les logs :

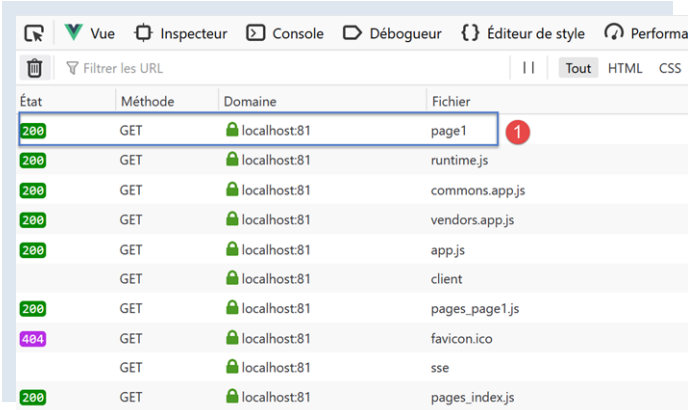
- en **[1]**, le serveur a réexécuté la fonction **[nuxtServerInit]**. Or celle-ci est la suivante :

```
1.  /* eslint-disable no-console */
2.  export const state = () => ({
3.    // compteut
4.    counter: 0
5.  })
6.
7.  export const mutations = {
8.    // incrémentation du compteur d'une valeur [inc]
```

```
9.    increment(state, inc) {
10.     state.counter += inc
11.   }
12. }
13.
14. export const actions = {
15.   async nuxtServerInit(store, context) {
16.     // qui exécute ce code ?
17.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server)
18.     // on attend la fin d'une promesse
19.     await new Promise(function(resolve, reject) {
20.       // on a normalement ici une fonction asynchrone
21.       // on la simule avec une attente d'une seconde
22.       setTimeout(() => {
23.         // succès
24.         resolve()
25.       }, 1000)
26.     })
27.     // on modifie le store
28.     store.commit('increment', 53)
29.     // log
30.     console.log('nuxtServerInit commit terminé')
31.   }
32. }
```

La ligne 28 affecte la valeur 53 au compteur.

Examinons la requête HTTP faite par le navigateur lorsqu'on a rafraîchi la page **[page1]** :



On voit qu'outre la page **[page1]** **[1]**, le client demande un certain nombre de scripts au serveur. On notera les scripts **[pages_index, pages_page1]** qui sont les scripts associés aux pages **[index, page]**. Ces scripts sont fournis à chaque requête au serveur quelque soit la page demandée ;

En **[1]**, la page **[page1]** est demandée au serveur avec la requête HTTP suivante :

```
1.  GET http://localhost:81/nuxt-04/page1
2.  Host: localhost:81
3.  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
4.  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5.  Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
6.  Accept-Encoding: gzip, deflate
7.  DNT: 1
8.  Connection: keep-alive
9.  Upgrade-Insecure-Requests: 1
10. Pragma: no-cache
11. Cache-Control: no-cache
```

On voit que cette requête ne transmet aucune information au serveur. Celui-ci n'a donc aucun moyen de connaître l'état du store **[Vuex]** côté client. Il aurait fallu pour cela que le client lui envoie cette information.

Dans le projet suivant **[nuxt-05]** nous utilisons un cookie pour que le client puisse envoyer de l'information au serveur lorsque celui-ci est sollicité.

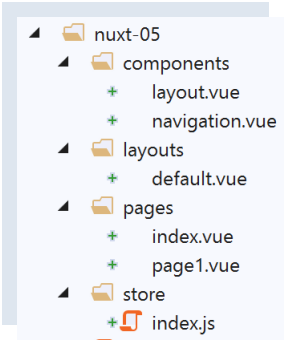
4.8 Exemple [nuxt-05] : persistance du store avec un cookie de session

Objectif : on voudrait que le store **[Vuex]** ne soit pas réinitialisé à chaque requête vers le serveur. Pour ce faire nous allons utiliser un cookie de session :

- le store sera initialisé par le serveur et mis par celui-ci dans un cookie de session ;
- le navigateur client recevra ce cookie de session et mécaniquement l'enverra à chaque nouvelle requête vers le serveur ;
- le serveur pourra alors récupérer ce cookie de session et travailler avec le store qu'il contient, un store mis à jour par le client ;

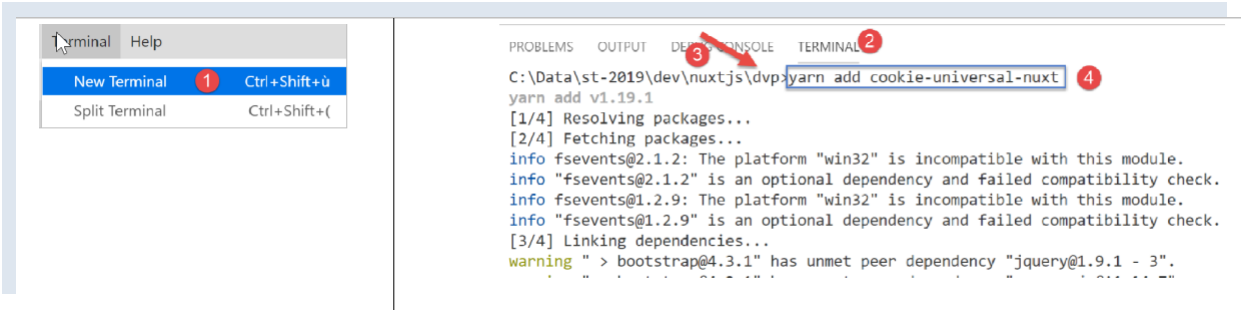
4.8.1 Présentation

Le projet **[nuxt-05]** est obtenu initialement par recopie du projet **[nuxt-04]** :



Nous allons voir que seul le fichier **[store / index.js]** va changer.

Pour utiliser des cookies avec **[nuxt]**, nous allons utiliser le module **[cookie-universal-nuxt]** que nous installlons avec **[yarn]** dans un terminal VSCode :



- en **[4]**, on tape la commande **[yarn add cookie-universal-nuxt]** ;

Un nouveau module est ainsi ajouté au fichier **[package.json]** du projet **[dvp]** :

```
1.  ...
2.  },
3.    "dependencies": {
4.      "@nuxtjs/axios": "^5.3.6",
5.      "bootstrap": "^4.1.3",
6.      "bootstrap-vue": "^2.0.0",
7.      "cookie-universal-nuxt": "^2.0.19",
8.      "nuxt": "^2.0.0"
9.  },
```

4.8.2 Le fichier de configuration [nuxt.config.js]

Pour que **[nuxt]** puisse utiliser les cookies de **[cookie-universal-nuxt]**, il faut déclarer ce module dans le fichier de configuration **[nuxt.config.js]** :

```
1.    /*
2.     ** Nuxt.js modules
3.     */
4.    modules: [
```

```
5.      // Doc: https://bootstrap-vue.js.org
6.      'bootstrap-vue/nuxt',
7.      // Doc: https://axios.nuxtjs.org/usage
8.      '@nuxtjs/axios',
9.      // https://www.npmjs.com/package/cookie-universal-nuxt
10.     'cookie-universal-nuxt'
11.   ],
```

- ligne 10, le module **[cookie-universal-nuxt]** est ajouté au tableau des modules **[6]** de **[nuxt]** ;

Le fichier **[nuxt.config.js]** est au final le suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: { color: '#fff' },
23.   /*
24.    ** Global CSS
25.    */
26.   css: [],
27.   /*
28.    ** Plugins to load before mounting the App
29.    */
30.   plugins: [],
31.   /*
32.    ** Nuxt.js dev-modules
33.    */
34.   buildModules: [
35.     // Doc: https://github.com/nuxt-community/eslint-module
36.     '@nuxtjs/eslint-module'
37.   ],
38.   /*
39.    ** Nuxt.js modules
40.    */
41.   modules: [
42.     // Doc: https://bootstrap-vue.js.org
43.     'bootstrap-vue/nuxt',
44.     // Doc: https://axios.nuxtjs.org/usage
45.     '@nuxtjs/axios',
46.     // https://www.npmjs.com/package/cookie-universal-nuxt
47.     'cookie-universal-nuxt'
48.   ],
49.   /*
50.    ** Axios module configuration
51.    ** See https://axios.nuxtjs.org/options
52.    */
53.   axios: {},
54.   /*
55.    ** Build configuration
56.    */
57.   build: {
58.     /*
59.      ** You can extend webpack config here
60.      */
61.     extend(config, ctx) {}
62.   },
63.   // répertoire du code source
64.   srcDir: 'nuxt-05',
```

```
65.   // routeur
66.   router: {
67.     // racine des URL de l'application
68.     base: '/nuxt-05/'
69.   },
70.   // serveur
71.   server: {
72.     // port de service, 3000 par défaut
73.     port: 81,
74.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
75.     // 0.0.0.0 = toutes les adresses réseau de la machine
76.     host: 'localhost'
77.   },
78.   // environnement
79.   env: {
80.     maxAge: 60 * 5
81.   }
82. }
```

- ligne 79 : on a ajouté la clé **[env]** au fichier. Cette clé est un mot réservé. Les éléments déclarés dans cet objet sont disponibles à partir de l'objet **[context.env]** dans les éléments de l'application ;
- ligne 80 : l'attribut **[maxAge]** sera la durée de vie maximale du cookie de session, durée qui se mesure à partir de la dernière fois que le cookie a été initialisé. Cette durée s'exprime en secondes. On a mis ici une durée de vie de 5 minutes ;

4.8.3 Le principe de la persistance du store

Les cookies échangés entre le client et le serveur sont disponibles de chaque côté (client et serveur) dans :

- **[context.app.$cookies]** là ou l'objet **[context]** est disponible, ç-à-d à peu près partout ;
- **[this.$cookies]** à l'intérieur d'une vue ;

On obtient un cookie particulier avec l'expression **[...$cookies.get('nom_du_cookie')]**. On fixe la valeur d'un cookie avec l'expression **[...$cookies.set('nom_du_cookie', valeur_du_cookie)]**.

Le principe du cookie de persistance du store sera le suivant :

- lorsque le serveur va initialiser le store dans la fonction **[nuxtServerInit]**, l'état du store sera stocké dans un cookie nommé 'session' ;
- le cookie 'session' fera alors partie de la réponse HTTP du serveur. On sait qu'un navigateur renvoie au serveur les cookies que celui-ci lui a envoyés. Il le fait à chaque nouvelle requête qu'il fait au serveur. On sait également que le serveur envoie le store à l'intérieur la page qu'il envoie au client ;
- au sein du navigateur, l'application cliente récupère le store envoyé par le serveur et va ensuite faire son travail. On fera en sorte qu'à chaque fois qu'elle modifie le store, le nouvel état de celui-ci soit stocké dans le cookie 'session' enregistré par le navigateur ;
- si l'utilisateur force un appel au serveur, le navigateur client renverra automatiquement tous les cookies que le serveur lui a précédemment envoyés, notamment le cookie nommé 'session' ;
- lorsque suite à cet appel, le serveur va réinitialiser de nouveau le store, il récupèrera le cookie nommé 'session' et initialisera l'état du store avec la valeur de celui-ci ;
- il y aura donc continuité du store entre le client et le serveur ;

4.8.4 Initialisation du store

Le store est implémenté dans le fichier **[store / index.js]** :

```
1.  /* eslint-disable no-console */
2.  export const state = () => ({
3.    // compteur
4.    counter: 0
5.  })
6.
7.  export const mutations = {
8.    // incrémentation du compteur d'une valeur [inc]
9.    increment(state, inc) {
10.     state.counter += inc
11.   },
12.   // remplacement du state
13.   replace(state, newState) {
14.     for (const attr in newState) {
15.       state[attr] = newState[attr]
16.     }
17.   }
18. }
```

```
19.
20. export const actions = {
21.   async nuxtServerInit(store, context) {
22.     // qui exécute ce code ?
23.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server, 'env=', context.env)
24.     // on attend la fin d'une promesse
25.     await new Promise(function(resolve, reject) {
26.       // on a normalement ici une fonction asynchrone
27.       // on la simule avec une attente d'une seconde
28.       setTimeout(() => {
29.         // init session
30.         initStore(store, context)
31.         // succès
32.         resolve()
33.       }, 1000)
34.     })
35.   }
36. }
37.
38. function initStore(store, context) {
39.   // y-a-t-il un cookie de session dans la requête en cours
40.   const cookies = context.app.$cookies
41.   const session = cookies.get('session')
42.   if (!session) {
43.     // pas de session existante
44.     console.log("nuxtServerInit, initialisation d'une nouvelle session")
45.     // on initialise le store
46.     store.commit('increment', 77)
47.   } else {
48.     console.log("nuxtServerInit, reprise d'une session existante")
49.     // on met à jour le store avec le cookie de session
50.     store.commit('replace', session.store)
51.   }
52.   // on met le store dans le cookie de session
53.   cookies.set('session', { store: store.state }, { path: context.base, maxAge: context.env.maxAge })
54.   // log
55.   console.log('initStore terminé, store=', store.state)
56. }
```

Commentaires

- lignes 2-5 : le store sera constitué d'un compteur ;
- lignes 9-11 : ce compteur pourra être incrémenté ;
- lignes 13-17 : l'état du store pourra être initialisé à partir d'un nouvel état. Cette fonction est là pour montrer une initialisation possible du store lorsque celui-ci n'est pas limité au seul compteur comme ici ;
- lignes 21-35 : la fonction **[nuxtServerInit]** n'a pas changé ;
- ligne 30 : lorsque le temps d'attente d'une seconde est écoulé, on initialise le store à l'aide de la fonction des lignes 38-56 ;
- lignes 40-41 : on commence par récupérer le cookie nommé 'session' :
  - lors de la 1ère exécution de l'application et lors de la 1ère requête faite au serveur, ce cookie **n'existera pas encore**. Il sera alors créé (ligne 53) et sera envoyé au navigateur client ;
  - lors de la même exécution de l'application et lors des requêtes n°s 2, 3, ... faites au serveur, ce cookie **existera** car le navigateur client le renverra avec chaque nouvelle requête faite au serveur ;
  - lors d'une seconde exécution de l'application et lors de la 1ère requête faite au serveur, ce cookie **peut exister également**. En effet, à l'issue de l'étape 1, le cookie a été stocké sur le navigateur avec une certaine durée de vie. Si cette durée de vie n'est pas dépassée, le cookie nommé 'session' sera envoyé avec la 1ère requête faite au serveur

  - En résumé, pour chaque requête faite au serveur : si le cookie 'session' est déjà stocké sur le navigateur client alors le serveur va le recevoir, sinon il ne le recevra pas.
- lignes 42-47 : si le serveur ne reçoit pas le cookie de session, alors le store est initialisé par la ligne 46 ;
  - puis ligne 53, un cookie nommé 'session' sera créé et placé dans la réponse HTTP du serveur. La valeur du cookie est l'objet **[{ store: store.state }]**. C'est donc l'état du store et non le store lui-même qui est mis dans le cookie de session ;
  - le 3ième paramètre de la fonction **[set]** est un objet d'options :
    - **[path]** indique à quelle URL ce cookie devra être renvoyé. **[context.base]** est l'URL de base de l'application **[nuxt-05]**. Celle-ci est définie dans le fichier **[nuxt.config.js]** :

```
  // routeur
  router: {
    // racine des URL de l'application
    base: '/nuxt-05/'
},
```

- **[maxAge]** est la durée de vie en secondes du cookie sur le navigateur. Passée cette durée, le navigateur ne le renvoie plus au serveur. **[context.env.maxAge]** renvoie là encore une valeur inscrite dans le fichier **[nuxt.config.js]** :

```
// environnement
  env: {
    maxAge: 60 * 5
}
```

- **[env]** est un mot clé réservé du fichier de configuration. On fixe ici la durée de vie à 5 minutes. Cette durée est mesurée par rapport à la dernière fois où le navigateur a reçu le cookie de session. Passée cette durée, le cookie ne sera pas renvoyé au serveur qui devra alors démarrer une nouvelle session ;

- lignes 48-50 : si le serveur reçoit le cookie de session, alors l'état du store est initialisé avec l'objet **[store]** du cookie de session. On se rappelle que cet objet contient l'état sauvegardé du store ;
- puis ligne 53, le cookie de session sera placé dans la réponse faite au navigateur client :
  - la fonction **[get]** va chercher le cookie de session dans la **requête reçue** par le serveur ;
  - la fonction **[set]** met le cookie de session dans la **réponse** que le serveur fait au navigateur client ;

4.8.5 Incrémentation du compteur du store

L'incrémentation du compteur dans la page **[index.vue]** évolue de la façon suivante :

```
1.    // gestion des évts
2.    methods: {
3.      incrementCounter() {
4.        console.log('incrementCounter')
5.        // incrément du compteur de 1
6.        this.$store.commit('increment', 1)
7.        // chgt de la valeur affichée
8.        this.value = this.$store.state.counter
9.        // sauvegarde du store dans le cookie de session
10.       this.$cookies.set('session', { store: this.$store.state }, { path: this.$nuxt.context.base, maxAge: t
    his.$nuxt.context.env.maxAge })
11.     }
12.   }
 1.
```

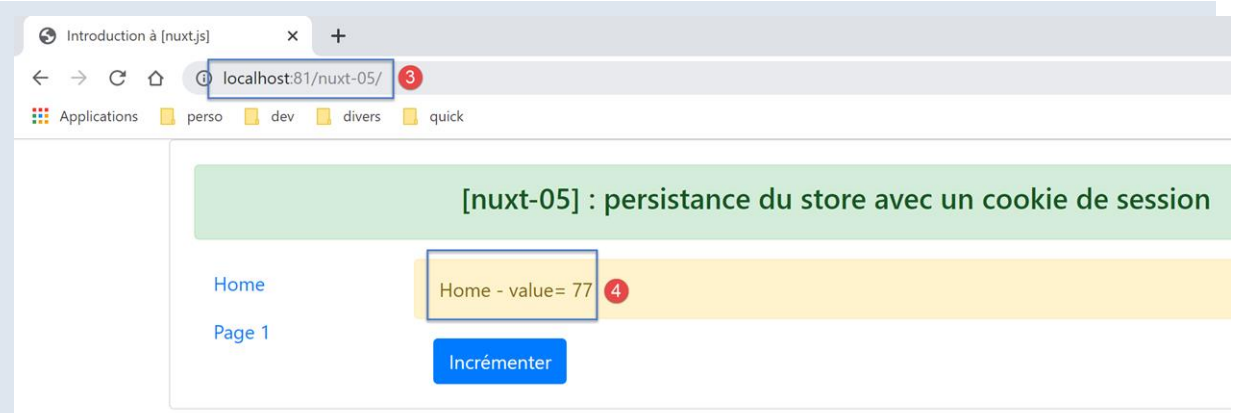
Du côté client, à chaque fois qu'on modifie le store, il faut le sauvegarder dans le cookie de session. En effet, l'utilisateur peut demander une URL à la main à tout moment et on doit être alors capable d'envoyer au serveur un store à jour. C'est pourquoi, ligne 10, après l'incrémentation du compteur du store, on sauvegarde l'état de celui-ci dans le cookie de session :

- les cookies sont disponibles dans la propriété **[this.$cookies]** ;
- l'état du store **[this.$store.state]** est sauvegardé dans le cookie associé à la clé **[store]** ;
- le chemin du cookie est **[context.base]**. Dans une vue, le contexte est disponible dans **[this.$nuxt.context]** ;
- la durée de vie du cookie est **[context.env.maxAge]** disponible ici dans la propriété **[this.$nuxt.context.env.maxAge]** ;

4.8.6 Exécution de l'exemple [nuxt-05]

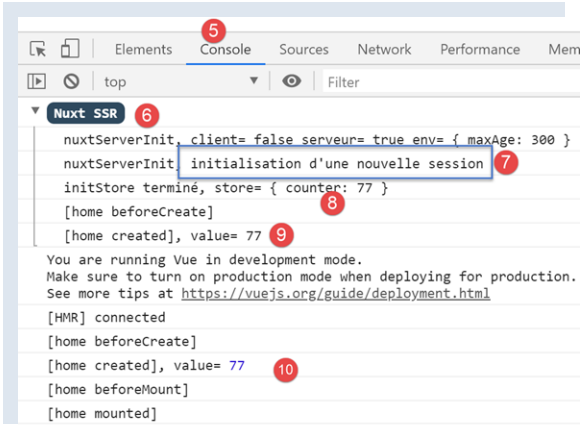
Nous lançons l'application **[nuxt-05]** :

Les copies d'écran qui suivent sont celles d'un navigateur Chrome. Nous demandons l'URL **[http://localhost:81/nuxt-05/]**. **N'oubliez pas** le dernier / derrière /nuxt-05, sinon vous n'aurez pas les résultats escomptés :



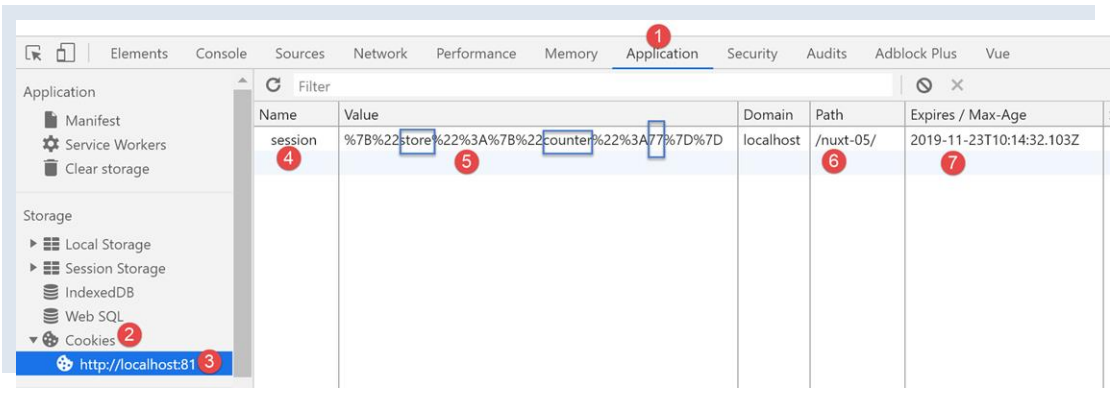
- en **[4]**, nous avons obtenu la valeur initiale du store (77) ;

Examinons les logs du navigateur (F12) :



- en **[5-6]**, les logs du serveur ;
- en **[7]**, on voit que le serveur démarre une nouvelle session. Cela veut dire qu'il n'a pas reçu de cookie de session ;
- en **[8]**, initialisation du compteur avec la valeur 77 ;
- en **[9]**, la page **[index]** du serveur (9) et celle du client (10) affichent bien la même valeur du compteur ;

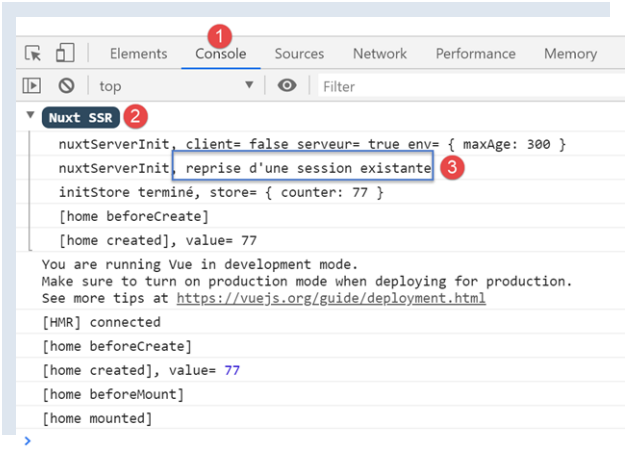
Maintenant regardons les cookies reçus par le navigateur :



- en **[1]**, choisissez l'onglet **[Application]** puis l'option **[Cookies]** **[2]**. Parmi tous les cookies de votre navigateur, choisissez celui du domaine **[http://localhost:81]**;
- en **[4]**, le cookie nommé 'session'. Si vous ne l'avez pas, rechargez la page **[F5]** : peut-être avez-vous dépassé sa durée de vie qui est de 5 mn ;
- en **[5]**, la valeur du cookie. Bien que ce ne soit pas très lisible à cause de l'encodage des caractères { :, on distingue la valeur 77 du compteur ;

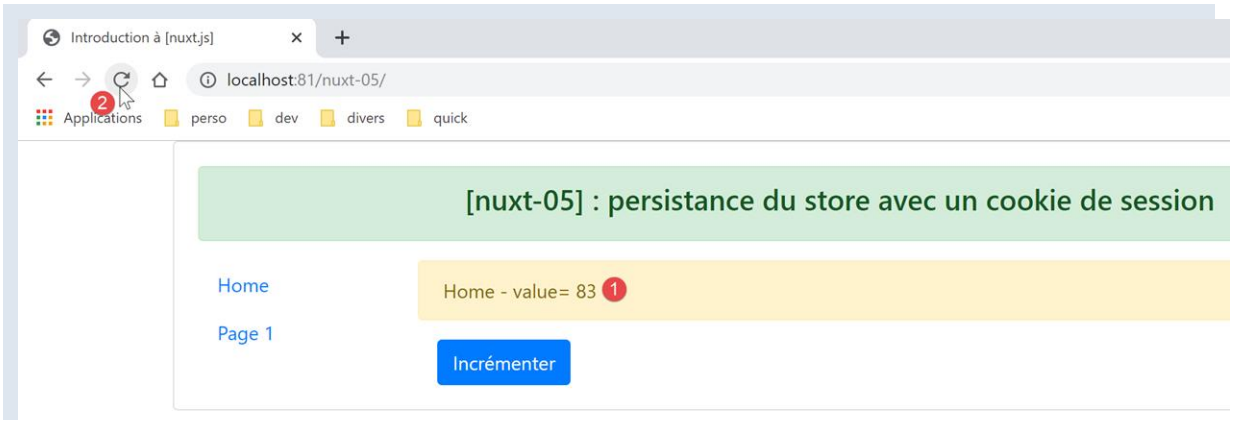
- en **[6]**, l'URL du cookie : à chaque fois que cette URL sera demandée, le navigateur enverra le cookie au serveur ;
- en **[7]**, l'heure de fin de validité du cookie. Lorsque cette heure sera dépassée, le cookie sera détruit sur le navigateur ;

Assurez-vous d'avoir ce cookie. Si vous ne l'avez pas, rechargez la page (F5). Lorsque vous avez la page avec son cookie, rechargez de nouveau la page (F5). Les logs deviennent alors les suivants :

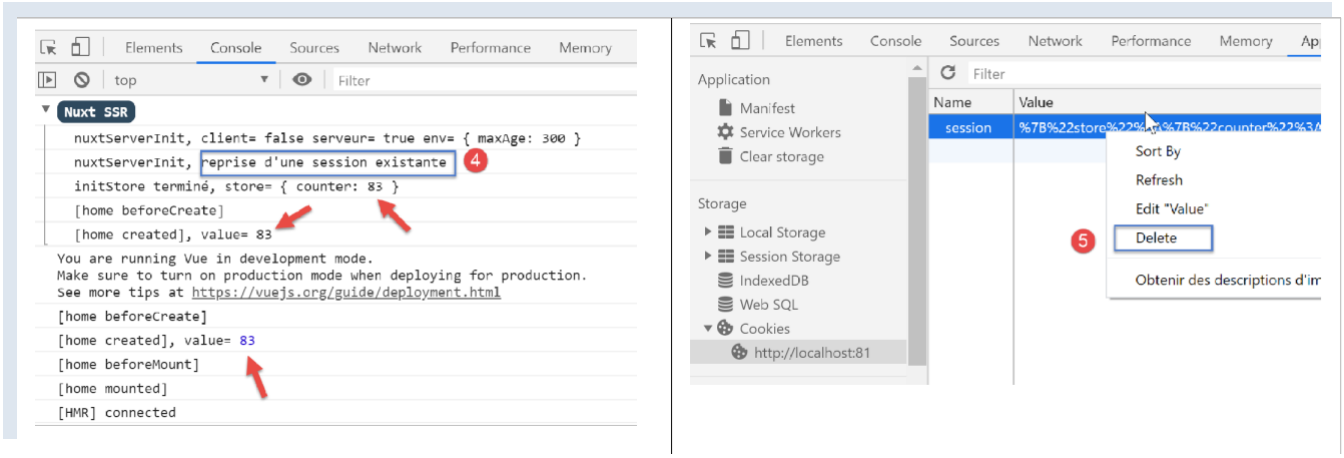


Cette fois-ci en **[3]**, le serveur a bien récupéré le cookie de session. C'est le navigateur client qui le lui a envoyé.

Maintenant, procédez à des incrémentations du compteur, puis de temps en temps rechargez la page courante (F5), que ce soit **[index]** ou **[page1]**, vous devez constater que le compteur ne revient pas à 77 comme dans l'exemple **[nuxt-04]** mais garde la valeur qu'il avait sur le navigateur client avant le rechargement de la page :



Les logs du navigateur sont alors les suivants :



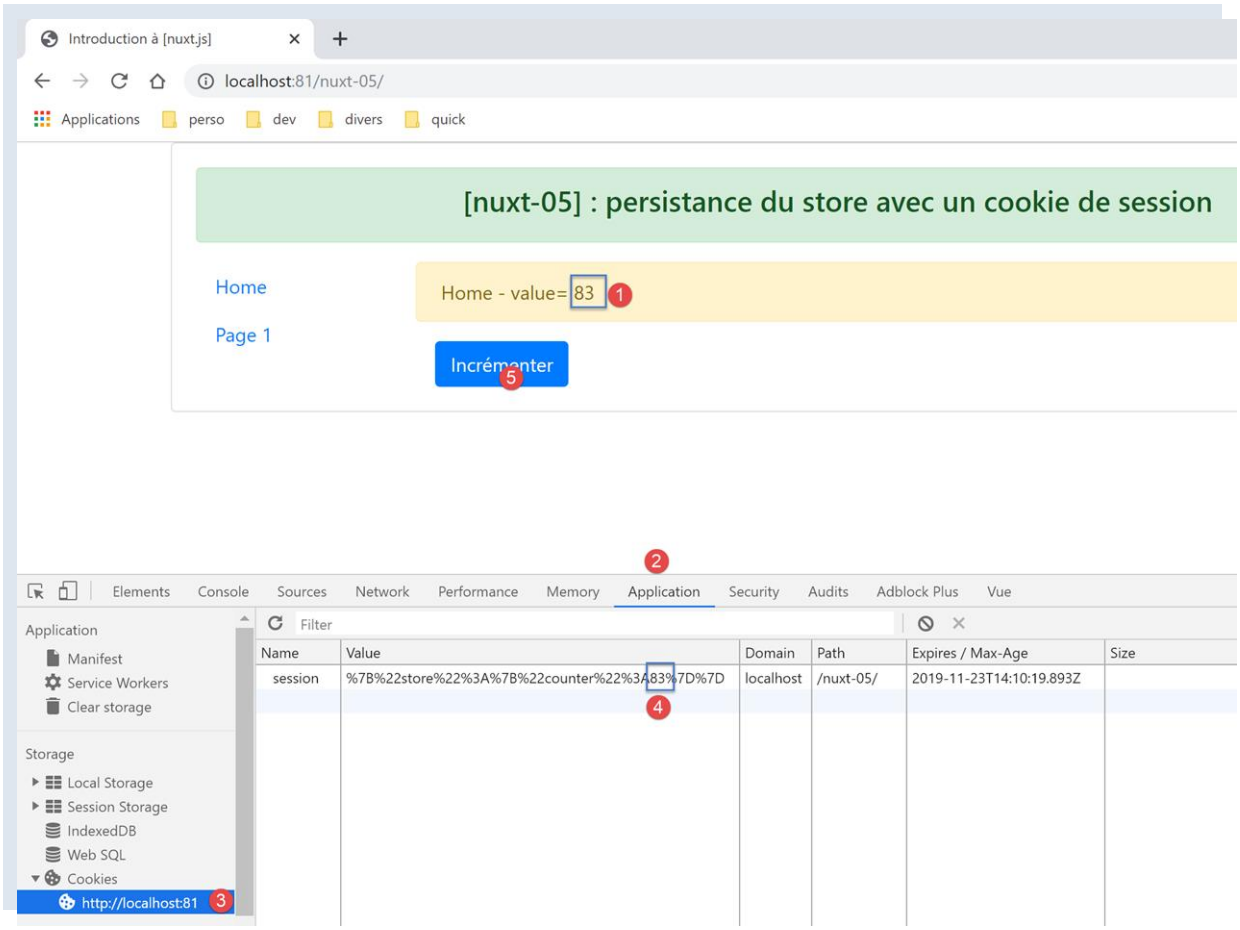
Note : pour les tests vous pouvez avoir besoin de supprimer **[5]** le cookie de session stocké sur le navigateur pour repartir avec une nouvelle session, initialisée par le serveur, lors de la prochaine requête vers celui-ci.

Enfin montrons l'influence de la fonction **[incrementCounter]** de la page **[index]** sur le cookie de session stocké sur le navigateur client :

```
1.    // gestion des évts
2.    methods: {
3.      incrementCounter() {
4.        console.log('incrementCounter')
5.        // incrément du compteur de 1
6.        this.$store.commit('increment', 1)
7.        // chgt de la valeur affichée
8.        this.value = this.$store.state.counter
9.        // sauvegarde du store dans le cookie de session
10.       this.$cookies.set('session', { store: this.$store.state }, { path: this.$nuxt.context.base, maxAge: this.$nuxt.contex
    t.env.maxAge })
11.     }
12.   }
```

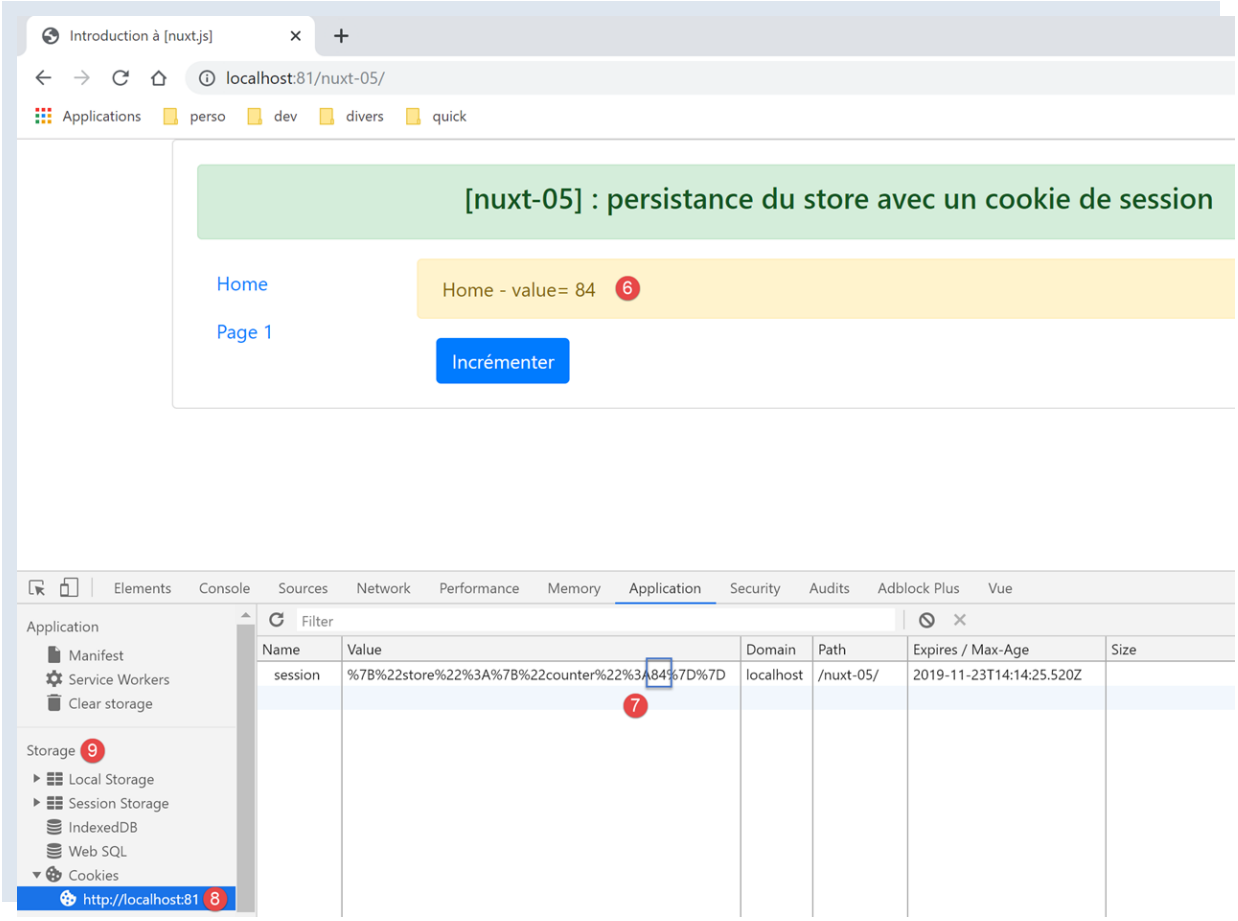
- ligne 10 : la modification du compteur est répercutée sur le cookie de session ;

Vérifions ce point. On part de la situation suivante :



- en **[4]**, le compteur du cookie de session reflète bien la valeur affichée **[1]** ;

Maintenant, incrémentons le compteur une fois **[5]**. Le cookie de session en **[4]** évolue de la façon suivante :



- en **[7]**, le compteur du cookie de session est bien passé à 84. Pour le voir, il faut rafraîchir la vue **[8]**. Pour cela sélectionnez une autre option du **[Storage] [9]**, puis resélectionnez l'option **[8]**. La nouvelle valeur du cookie de session devrait alors apparaître ;

4.9 Exemple [nuxt-06] : injection dans le contexte [nuxt] d'un gestionnaire de session

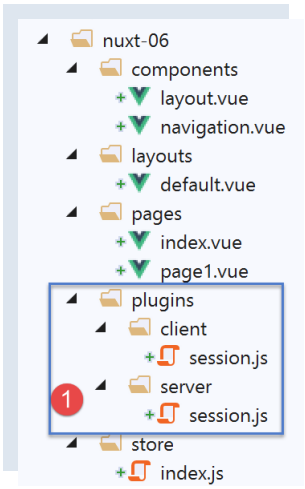
4.9.1 Présentation

L'exemple **[nuxt-05]** a montré qu'on pouvait persister le store même lorsque l'utilisateur force des appels au serveur. Les éléments du store sont réactifs pour que s'ils sont intégrés à des vues celles-ci soient réactives aux changements du store. On peut vouloir aussi persister des éléments au fil des échanges client / serveur sans pour autant vouloir qu'ils soient réactifs, tout simplement parce qu'ils ne sont pas affichés par des vues. On peut alors stocker ceux-ci dans la session sans qu'ils soient pour autant dans le store.

Le store est accessible facilement au travers de propriétés telles que **[context.app.$store]** en-dehors des vues ou **[this.$store]** dans les vues. On voudrait quelque chose d'analogue pour la session, quelque chose comme **[context.app.$session]** ou **[this.$session]**. On va voir que c'est possible grâce au concept d'injection. Seulement on ne peut pas injecter des objets dans le contexte, seulement des fonctions. Celle-ci sera alors disponible au travers des expressions **[context.app.$session()]** ou **[this.$session()]**.

Enfin, nous allons présenter le concept **[nuxt]** de **[plugin]**.

L'exemple **[nuxt-06]** est obtenu initialement par recopie du projet **[nuxt-05]** :



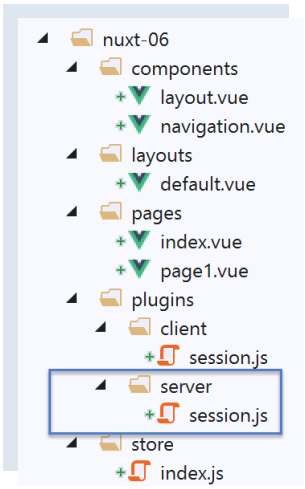
- en **[1]**, nous ajouterons un dossier **[plugins]** ;

4.9.2 la notion de plugin [nuxt]

[nuxt] nomme **[plugin]** tout code exécuté au démarrage de l'application, avant même l'exécution de la fonction **[nuxtServerInit]** par le serveur qui était jusqu'à maintenant la première fonction utilisateur à être exécutée. Les plugins de l'application doivent être déclarés dans la clé **[plugins]** du fichier de configuration **[nuxt.config.js]** :

```
1.  /*
2.     ** Plugins to load before mounting the App
3.     */
4.    plugins: [
5.      { src: '~/plugins/client/session', mode: 'client' },
6.      { src: '~/plugins/server/session', mode: 'server' }
7.    ],
```

- lignes 5-6 : un plugin est désigné par son chemin **[src]** et son mode d'exécution **[mode]**. **[mode]** peut avoir trois valeurs :
  - **[client]** : le plugin doit être exécuté côté client uniquement ;
  - **[server]** : le plugin doit être exécuté côté serveur uniquement ;
  - absence de la clé **[mode]** : dans ce cas, le plugin doit être exécuté à la fois côté client et serveur ;
- lignes 5-6 : nous avons placé nos deux plugins dans un dossier **[plugins]**. Il n'y a aucune obligation à cela. Les plugins peuvent être placés n'importe où dans l'arborescence du projet. De même les noms des sous-dossiers **[client, server]** sont ici arbitraires ;



4.9.3 Le plugin [session] du serveur

Le plugin **[server / session.js]** est le suivant :

```
1.  /* eslint-disable no-console */
2.  export default (context, inject) => {
```

```
3.    // gestion de la session serveur
4.
5.    // y-a-t-il une session existante ?
6.    let value = context.app.$cookies.get('session')
7.    if (!value) {
8.      // nouvelle session
9.      console.log("[plugin session server], démarrage d'une nouvelle session")
10.     value = initValue
11.   } else {
12.     // session existante
13.     console.log("[plugin session server], reprise d'une session existante")
14.   }
15.   // définition de la session
16.   const session = {
17.     // contenu de la session
18.     value,
19.     // sauvegarde de la session dans un cookie
20.     save(context) {
21.       context.app.$cookies.set('session', this.value, { path: context.base, maxAge: context.env.maxAge })
22.     }
23.   }
24.   // on injecte une fonction dans [context, Vue] qui rendra la session courante
25.   inject('session', () => session)
26. }
27.
28. // valeur initiale de la session
29. const initValue = {
30.   initSessionDone: false
31. }
```

- ligne 2 : les plugins sont exécutés à chaque fois qu'il y a un appel au serveur : au démarrage et à chaque fois que l'utilisateur force un appel au serveur en tapant une URL à la main :
  - c'est d'abord le (ou les) plugin(s) du serveur qui est (sont) exécuté(s) en premier ;
  - lorsque le navigateur client a reçu la réponse du serveur, c'est au tour du (ou des) plugin(s) du client de s'exécuter ;
- ligne 2 : tout plugin, client ou serveur, reçoit deux paramètres :
  - **[context]** : le contexte du serveur ou du client selon qui exécute le plugin ;
  - **[inject]** : une fonction qui permet d'injecter une fonction dans le contexte du serveur ou du client ;
- le but du plugin **[server / session]** est double :
  - définir une session (lignes 16-23) ;
  - définir au sein du contexte, une fonction **[$session]** qui rendra comme résultat, la session de la ligne 16. C'est la ligne 25 qui fait cela ;
- lignes 16-23 : la session encapsulera ses données dans l'objet **[value]** de la ligne 18 ;
- lignes 20-22 : elle dispose d'une fonction **[save]** qui reçoit en paramètre un objet **[context]**. C'est le code appelant qui lui fournit ce contexte. Avec celui-ci, la fonction **[save]** sauvegarde la valeur de la session, l'objet **[value]**, dans le cookie de session ;
- ligne 6 : lorsque le plugin **[server / session]** s'exécute, il commence par regarder si le serveur a reçu un cookie de session ;
  - si oui, l'objet **[value]** de la ligne 6 représente la valeur de la session, l'ensemble des données encapsulées dans celle-ci ;
  - si non, lignes 7-11, on fixe la valeur initiale de la session. Celle-ci sera l'objet **[initValue]** des lignes 29-31. Les éléments de la session seront définis dans la fonction **[nuxtServerInit]** qui est exécutée après le plugin serveur ;
- ligne 18 : la notation **[value]** est un raccourci pour la notation **[value:value]**. Le **[value]** de gauche est le nom d'une clé d'objet, le **[value]** de droite est l'objet **[value]** déclaré ligne 6 ;
- ligne 25 : lorsqu'on arrive à cette ligne, la session a été soit créée parce qu'elle n'existait pas, soit récupérée dans la requête HTTP du navigateur client ;
- ligne 25 : on injecte dans le contexte serveur une nouvelle fonction :
  - le 1er paramètre de **[inject]** est le nom de la fonction qu'on crée, ici 'session'. **[nuxt]** lui donnera en fait le nom '**$**session' ;
  - le 2ième paramètre est la définition de la fonction. Ici la fonction **[$session]**
    - n'admettra aucun paramètre ;
    - rendra l'objet **[session]** de la ligne 16 ;
- une fois le plugin exécuté :
  - la fonction **[$session]** est disponible dans **[context.app.$session]** là où l'objet **[context]** est disponible, ou **[this.$session]** dans une vue ou dans le store **[vuex]** ;
  - la fonction **[$session]** rend un objet **[session]** avec une unique clé **[value]** ;
  - à la création initiale de la session, l'objet **[value]** n'a qu'une clé **[initStoreDone]** (lignes 29-31). La clé **[initStoreDone:false]** sert à indiquer que le store n'a pas encore été placé dans la session. Cela va être fait par la fonction **[nuxtServerInit]** ;

4.9.4 Initialisation de la session

Une fois le plugin **[session / server]** exécuté par le serveur, celui-ci va exécuter le script **[store / index.js]** suivant :

```
1.  /* eslint-disable no-console */
2.  export const state = () => ({
3.    // compteur
4.    counter: 0
5.  })
6.
7.  export const mutations = {
8.    // incrémentation du compteur d'une valeur [inc]
9.    increment(state, inc) {
10.     state.counter += inc
11.   },
12.   // remplacement du state
13.   replace(state, newState) {
14.     for (const attr in newState) {
15.       state[attr] = newState[attr]
16.     }
17.   }
18. }
19.
20. export const actions = {
21.   async nuxtServerInit(store, context) {
22.     // qui exécute ce code ?
23.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server, 'env=', context.env)
24.     // on attend la fin d'une promesse
25.     await new Promise(function(resolve, reject) {
26.       // on a normalement ici une fonction asynchrone
27.       // on la simule avec une attente d'une seconde
28.       setTimeout(() => {
29.         // init session
30.         initSession(store, context)
31.         // succès
32.         resolve()
33.       }, 1000)
34.     })
35.   }
36. }
37.
38. function initSession(store, context) {
39.   // store est le store à initialiser
40.
41.   // on récupère la session
42.   const session = context.app.$session()
43.   // la session a-t-elle été déjà initialisée ?
44.   if (!session.value.initSessionDone) {
45.     // on démarre un nouveau store
46.     console.log("nuxtServerInit, initialisation d'une nouvelle session")
47.     // on initialise le store
48.     store.commit('increment', 77)
49.     // on met le store dans la session
50.     session.value.store = store.state
51.     // on initialise une nouvelle session
52.     session.value.somethingImportant = { x: 2, y: 4 }
53.     // la session est désormais initialisée
54.     session.value.initSessionDone = true
55.   } else {
56.     console.log("nuxtServerInit, reprise d'un store existant")
57.     // on met à jour le store avec le store de la session
58.     store.commit('replace', session.value.store)
59.   }
60.   // on sauvegarde la session
61.   session.save(context)
62.   // log
63.   console.log('initSession terminé, store=', store.state, 'session=', session.value)
64. }
```

Par rapport au store du projet **[nuxt-05]**, seule la fonction **[initSession]** (anciennement *initStore*) des lignes 38-60 change :

- ligne 42 : on récupère la session grâce à la fonction **[$session]** qui a été injectée dans le contexte du serveur ;
- ligne 44 : on regarde si la session a déjà été initialisée ;
- lignes 45-54 : si ce n'est pas le cas :
  - ligne 48 : le store est initialisé ;
  - ligne 50 : l'état du store est mis dans la session ;
  - ligne 52 : on ajoute un autre objet **[somethingImportant]** dans la session. Celui-ci ne fera pas partie du store ;
  - ligne 54 : on note le fait que la session est désormais initialisée ;

- lignes 55-59 : si la session était déjà initialisée :
  - ligne 58 : le nouveau store est initialisé avec le contenu de la session ;
- ligne 61 : la session est sauvegardée dans le cookie de session. On rappelle que cela consiste à placer le cookie dans la réponse HTTP que le serveur va faire au navigateur client ;

4.9.5 Le plugin `[client / session]` du client

Une fois que le serveur a exécuté les scripts **`[plugins / server / session]`** et **`[store / index]`**, il va envoyer l'une des pages **`[index, page1]`** au navigateur client. Dans la réponse HTTP du serveur, il y aura le cookie de session. Une fois la page reçue par le navigateur client, les scripts client embarqués dans la page vont s'exécuter. Le plugin **`[client / session]`** va alors s'exécuter :

```
1.  /* eslint-disable no-console */
2.  export default (context, inject) => {
3.    // gestion de la session client
4.
5.    // la session existe forcément, initialisée par le serveur
6.    console.log('[plugin session client], reprise de la session du serveur')
7.
8.    // définition de la session
9.    const session = {
10.     // contenu de la session
11.     value: context.app.$cookies.get('session'),
12.     // sauvegarde de la session dans un cookie
13.     save(context) {
14.       context.app.$cookies.set('session', this.value, { path: context.base, maxAge: context.env.maxAge })
15.     }
16.   }
17.
18.   // on injecte une fonction dans [context, Vue] qui rendra la session courante
19.   inject('session', () => session)
20. }
```

- lorsque le plugin client s'exécute, le cookie de session a déjà été reçu par le navigateur client ;
- l'objectif du plugin **`[client]`** est d'injecter lui-aussi une fonction **`[$session]`** dans le contexte du client. Cette fonction rendrait la session envoyée par le serveur ;
- ligne 19 : la fonction injectée **`[$session]`** rendra la session des lignes 9-16 ;
- lignes 9-16 : l'objet **`[session]`** géré par le client. Ce sera une copie de la session envoyée par le serveur ;
- ligne 11 : la valeur de la session client est prise dans le cookie de session envoyé par le serveur **`[nuxt]`** ;
- lignes 13-15 : comme pour la session du serveur, la session du client a une fonction **`[save]`** qui permet de sauvegarder la valeur de la session, **`[this.value]`** ligne 14, dans le cookie de session stocké sur le navigateur ;

4.9.6 La page `[index]`

La page **`[index]`** évolue de la façon suivante :

```
1.  <!-- page [index] -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <template slot="right">
8.        <b-
    alert show variant="warning"> Home - session= {{ jsonSession }}, counter= {{ $store.state.counter }} </b-
    alert>
9.        <!-- bouton -->
10.       <b-button @click="incrementCounter" class="ml-3" variant="primary">Incrémenter</b-button>
11.     </template>
12.   </Layout>
13. </template>
14.
15. <script>
16. /* eslint-disable no-undef */
17. /* eslint-disable no-console */
18. /* eslint-disable nuxt/no-env-in-hooks */
19.
20. import Layout from '@/components/layout'
21. import Navigation from '@/components/navigation'
22. export default {
23.   name: 'Home',
24.   // composants utilisés
25.   components: {
```

```
26.     Layout,
27.     Navigation
28.   },
29.   computed: {
30.     jsonSession() {
31.       return JSON.stringify(this.$session().value)
32.     }
33.   },
34.   // cycle de vie
35.   beforeCreate() {
36.     // client et serveur
37.     console.log('[home beforeCreate]')
38.   },
39.   created() {
40.     // client et serveur
41.     console.log('[home created], session=', this.$session().value)
42.   },
43.   beforeMount() {
44.     // client seulement
45.     console.log('[home beforeMount]')
46.   },
47.   mounted() {
48.     // client seulement
49.     console.log('[home mounted]')
50.   },
51.   // gestion des évts
52.   methods: {
53.     incrementCounter() {
54.       console.log('incrementCounter')
55.       // incrément du compteur de 1
56.       this.$store.commit('increment', 1)
57.       // modification session
58.       const session = this.$session()
59.       session.value.store = this.$store.state
60.       session.value.somethingImportant.x++
61.       session.value.somethingImportant.y++
62.       // sauvegarde de la session dans le cookie de session
63.       session.save(this.$nuxt.context)
64.     }
65.   }
66. }
67. </script>
```

Il faut se rappeler que cette page est exécutée aussi bien côté serveur que côté client.

- ligne 8 : on affiche maintenant et la session et le store ;
- ligne 30 : **[jsonSession]** est une propriété calculée qui rend la chaîne jSON de la valeur de la session ;
- ligne 41 : on affiche la valeur de la session à l'aide de la fonction injectée **[this.$session]**. Celle-ci existe aussi bien dans le contexte du serveur que dans celui du client ;
- ligne 53 : la méthode **[incrementCounter]** n'est elle exécutée que côté client ;
- ligne 56 : le compteur du store est incrémenté et affiché comme auparavant ;
- ligne 58 : on récupère la session grâce à la fonction injectée **[this.$session]** ;
- ligne 59 : le store de la session est mis à jour ;
- lignes 60-61 : on incrémente les attributs **[somethingImportant.x, somethingImportant.y]** de la session. Cela juste pour montrer qu'une session peut servir à transporter autre chose que le store ;
- ligne 63 : la session est sauvegardée dans le cookie de session stocké sur le navigateur. Dans une vue client, le contexte de celui-ci est disponible dans **[this.$nuxt.context]** ;

Le but de la page **[index]** est de montrer que la session n'est pas réactive alors que le store l'est. Lorsqu'on va incrémenter les éléments de la session, on découvrira que la vue n'est pas mise à jour. La vue **[page1]** présente une solution à ce problème.

4.9.7 La page [page1]

La page **[page1]** est obtenue par recopie de la page **[index]** puis quelque peu modifiée :

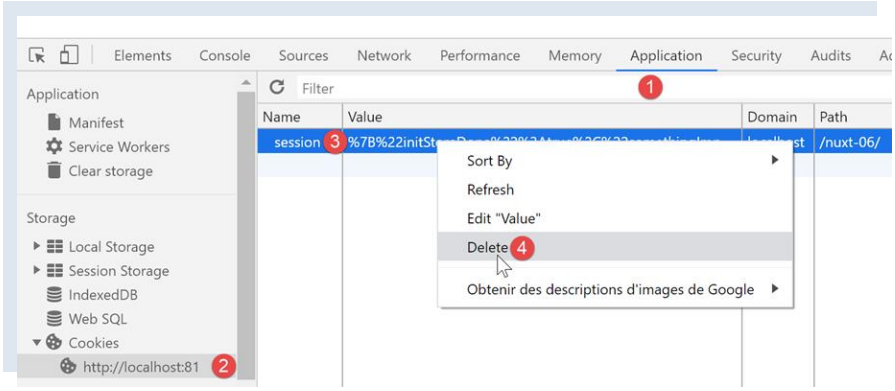
```
1.  <!-- page [index] -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <template slot="right">
8.        <b-alert show variant="warning"> Page1 - session= {{ jsonSession }}, counter= {{ $store.state.counter }} </b-alert>
```

```
9.        <!-- bouton -->
10.       <b-button @click="incrementCounter" class="ml-3" variant="primary">Incrémenter</b-button>
11.     </template>
12.   </Layout>
13. </template>
14.
15. <script>
16. /* eslint-disable no-undef */
17. /* eslint-disable no-console */
18. /* eslint-disable nuxt/no-env-in-hooks */
19.
20. import Layout from '@/components/layout'
21. import Navigation from '@/components/navigation'
22. export default {
23.   name: 'Page1',
24.   // composants utilisés
25.   components: {
26.     Layout,
27.     Navigation
28.   },
29.   data() {
30.     return {
31.       session: {}
32.     }
33.   },
34.   computed: {
35.     jsonSession() {
36.       return JSON.stringify(this.session.value)
37.     }
38.   },
39.   // cycle de vie
40.   beforeCreate() {
41.     // client et serveur
42.     console.log('[page1 beforeCreate]')
43.   },
44.   created() {
45.     // client et serveur
46.     // on met la session dans les propriétés réactives de la page
47.     this.session = this.$session()
48.     // log
49.     console.log('[page1 created], session=', this.session.value)
50.   },
51.   beforeMount() {
52.     // client seulement
53.     console.log('[page1 beforeMount]')
54.   },
55.   mounted() {
56.     // client seulement
57.     console.log('[page1 mounted]')
58.   },
59.   // gestion des évts
60.   methods: {
61.     incrementCounter() {
62.       console.log('incrementCounter')
63.       // incrément du compteur de 1
64.       this.$store.commit('increment', 1)
65.       // modification session
66.       this.session.value.store = this.$store.state
67.       this.session.value.somethingImportant.x++
68.       this.session.value.somethingImportant.y++
69.       // sauvegarde de la session dans le cookie de session
70.       this.session.save(this.$nuxt.context)
71.     }
72.   }
73. }
74. </script>
```

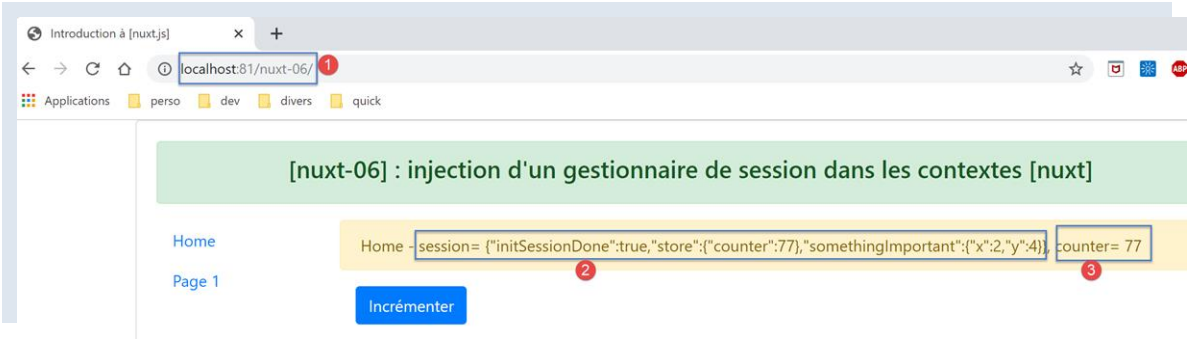
- ligne 47 : la principale différence vient du fait qu'on met la session courante dans les propriétés de la page (lignes 29-33). Cela va avoir pour effet que désormais la session va devenir réactive. Lorsque la fonction **`[incrementCounter]`** va incrémenter les éléments de la session, la vue **`[page1]`** sera mise à jour ;

4.9.8 Exécution du projet

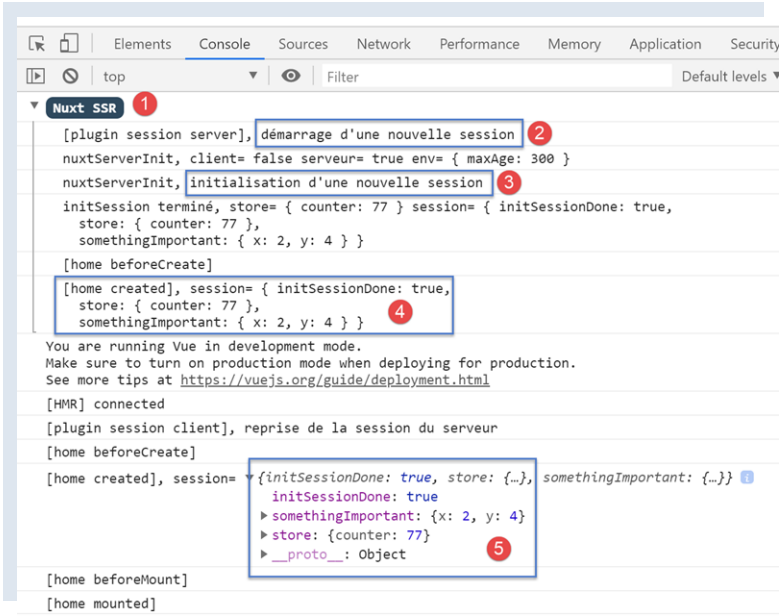
Avant d'exécuter le projet, vérifiez le cookie de session de votre navigateur et s'il existe supprimez-le pour que le serveur crée une session neuve :



Maintenant demandons l'URL **`[http://localhost:81/nuxt-06/]`** :

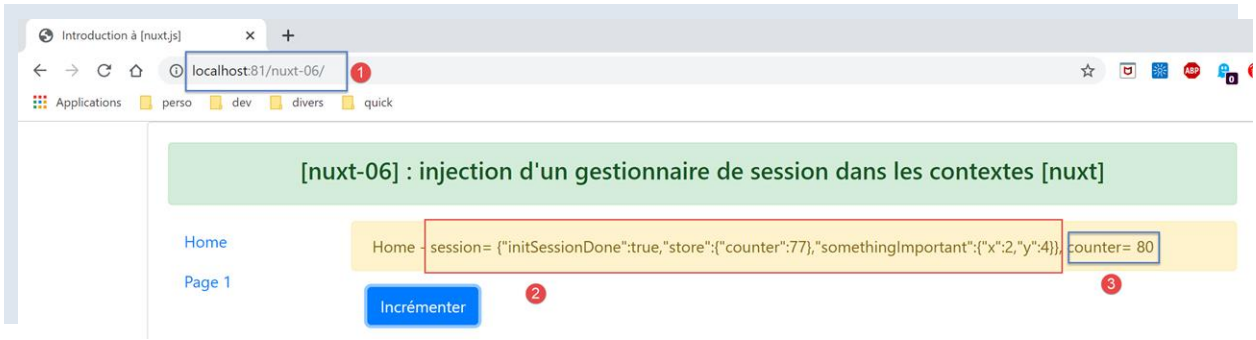


Les logs dans le navigateur sont alors les suivants :



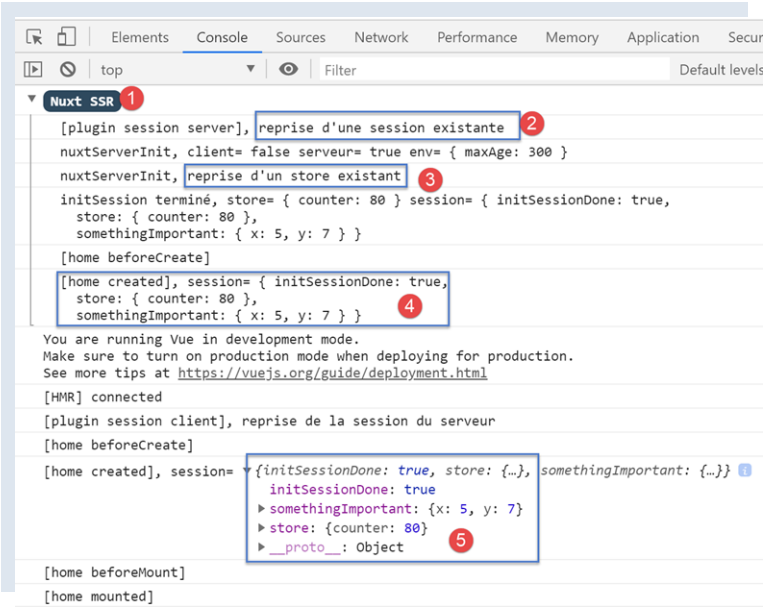
- en **`[2]`**, le serveur démarre une nouvelle session dans le plugin **`[session]`** du serveur ;
- en **`[3]`**, cette nouvelle session est initialisée dans **`[nuxtServerInit]`** ;
- en **`[4]`**, la nouvelle session telle qu'elle est connue sur le serveur ;
- en **`[5]`**, le client a correctement récupéré cette session ;

Maintenant incrémentons le compteur trois fois :



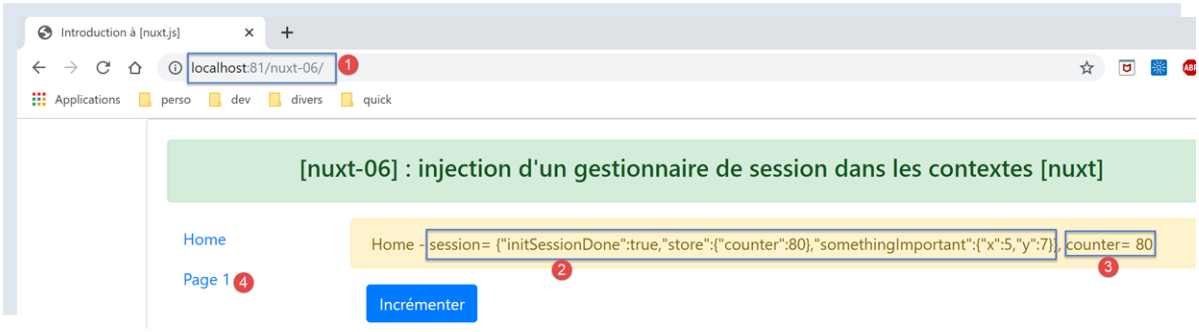
- en **[3]**, le compteur a bien été incrémenté mais pas la session en **[2]**. Alors que **[3]** affiche le store qui est réactif, **[2]** affiche la session qui elle n'est pas réactive :

Maintenant rechargeons la page (F5). Les logs sont les suivants suite à ce rechargement :



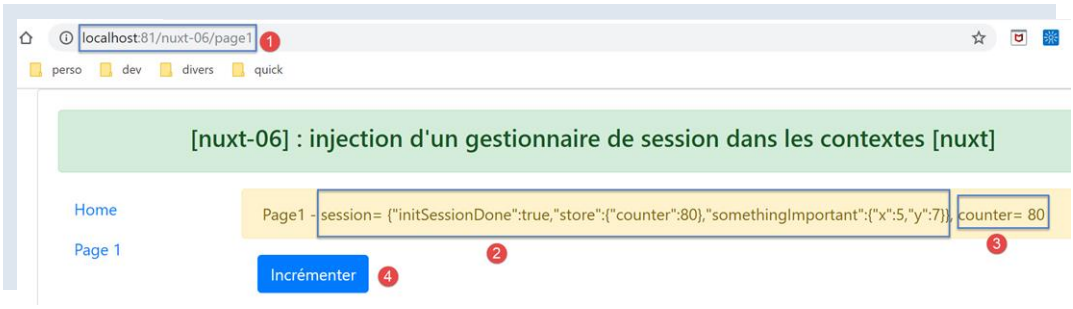
- en **[2]**, on voit que le serveur a reçu un cookie de session envoyé par le navigateur client ;
- en **[4]**, on voit que le store n'est pas réinitialisé mais repris dans la session reçue ;
- en **[4-5]** : on voit que les attributs de la session ont bien tous été incrémentés trois fois ;

La page envoyée par le serveur est alors la suivante ;

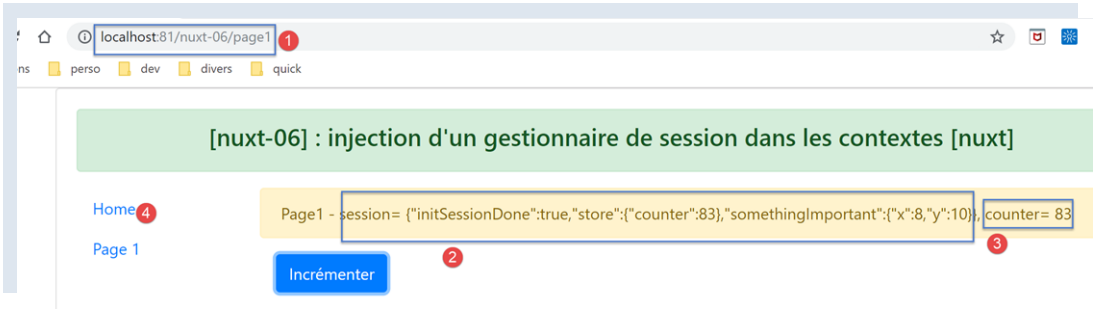


La conclusion issue de cette page est que la session peut transporter d'autres éléments que le store mais ceux-ci ne sont pas réactifs.

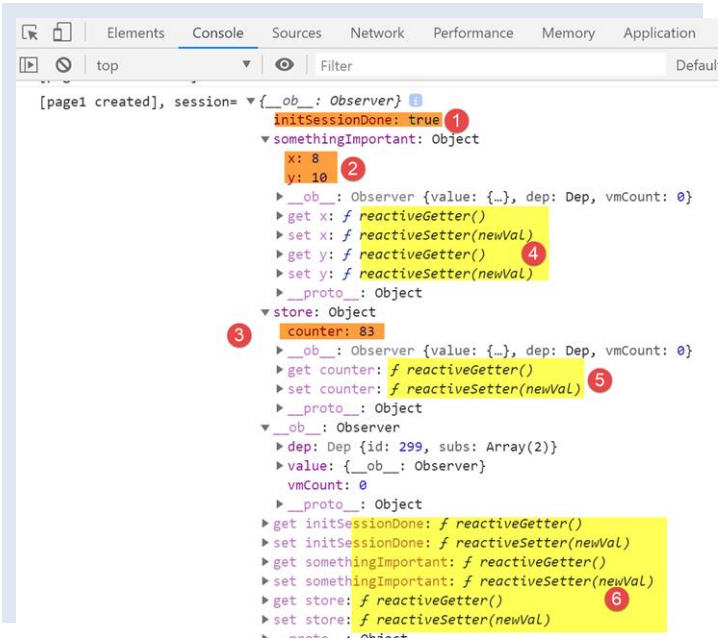
Maintenant cliquons sur le lien **[Page 1] [4]**. La nouvelle page affichée est alors la suivante :



Puis utilisons le bouton **[Incrémenter]** trois fois. La page devient la suivante :

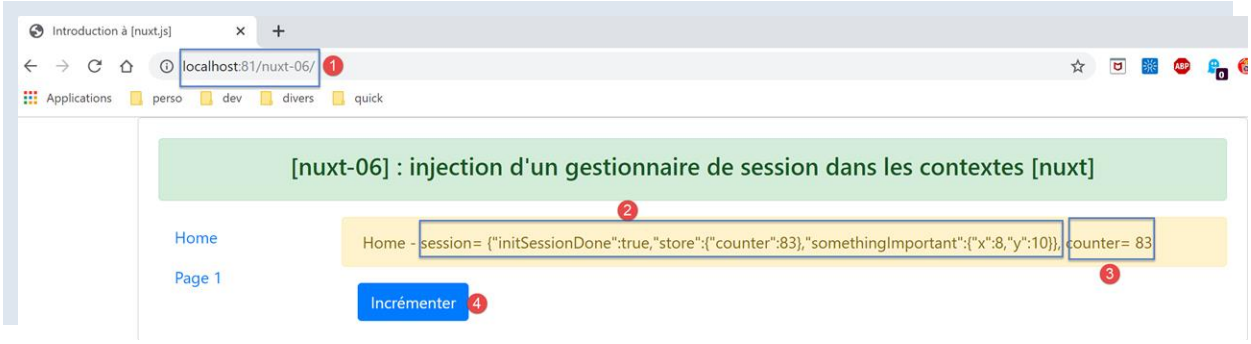


Cette fois-ci la session s'affiche correctement en **[2]**. Elle est ici réactive. Cela se voit dans les logs :

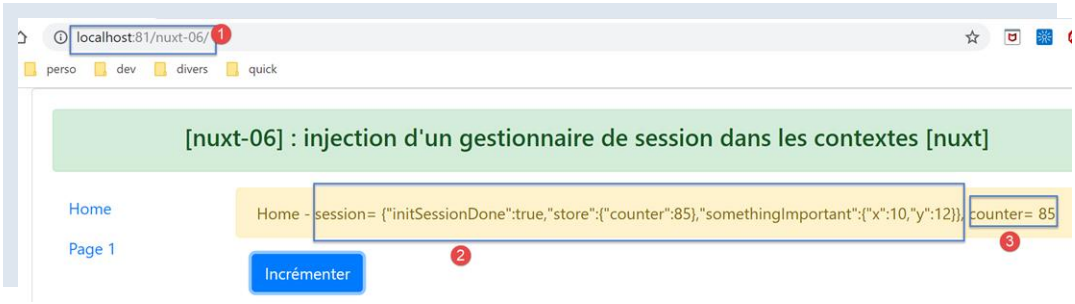


- en **[1-3]**, les valeurs de la session ;
- en **[4-6]**, les getters et setters réactifs des éléments de la session ;

Maintenant cliquons sur le lien **[Home] [4]**. Nous obtenons la page suivante :

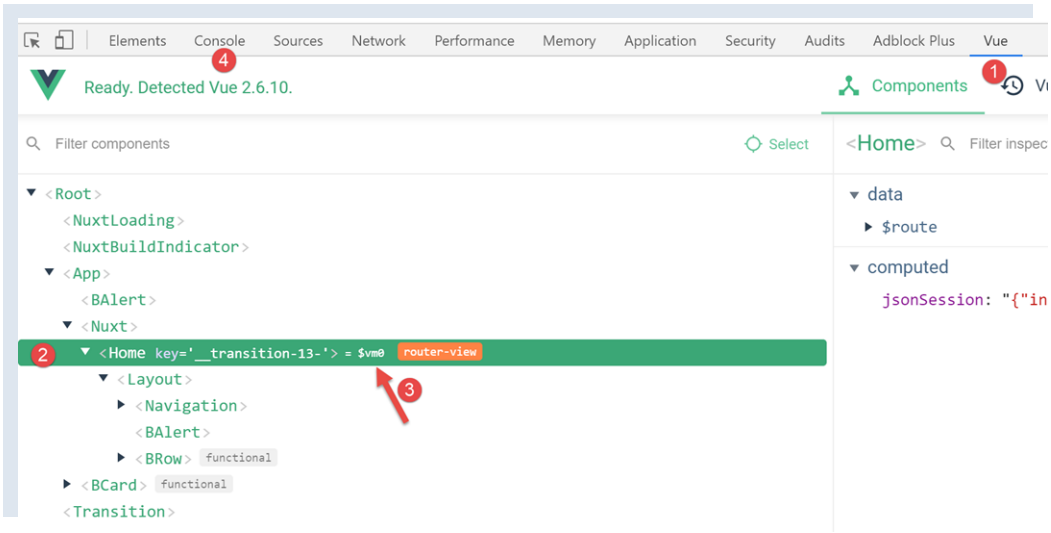


Puis cliquons deux fois sur le bouton **[Incrémenter] [4]**. La page devient la suivante :



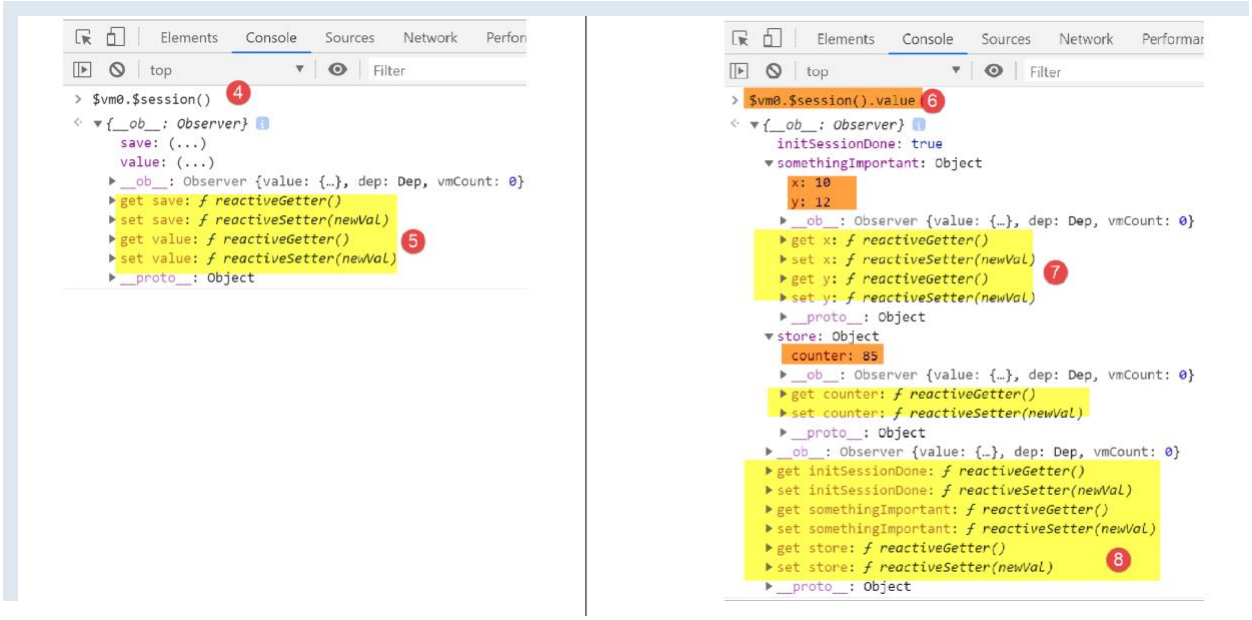
On constate que là également la session est devenue réactive **[2]**.

Demandons la valeur rendue par la fonction **[this.$session()]** :



- dans l'onglet **[Vue]**, on sélectionne la page courante **[Home]** pour en obtenir la référence **[$vm0] [3]** ;

Puis dans l'onglet **[Console] [4]**, demandons la valeur de la fonction **[$vm0.$session()]** :



- en **[5]**, on voit que la session est devenue réactive alors qu'elle ne l'était pas initialement ;
- en **[6]**, on demande à voir la valeur de la session ;
- en **[7-8]**, on découvre que cette valeur est elle également devenue réactive ;

On a donc là un résultat inattendu : si un élément devient réactif dans une page parce qu'il a été placé dans les propriétés de la page, alors il devient également réactif dans les pages où il ne fait pas partie des propriétés.

4.9.9 Conclusion

L'exemple **[nuxt-05]** a montré qu'on pouvait persister le store au fil des requêtes faites au serveur. L'exemple **[nuxt-06]** fait la même chose avec un objet qu'on a appelé **[session]** par analogie avec la session web. On a vu que cette session pouvait avoir les mêmes propriétés que le store **[Vuex]** et devenir réactive elle aussi alors que nativement elle ne l'était pas.

Alors quel est l'intérêt du store **[Vuex]** ? Je dois avouer que pour l'instant il ne m'est pas apparu. Il est probable que quelque chose m'a échappé. Donc dans le doute, je conseillerai d'utiliser :

- un **store [Vuex]** pour y mettre tout ce qui doit être partagé entre les pages du client, et ce qui éventuellement doit être partagé entre le client et le serveur ;
- un **cookie de session** si le store doit être persisté lors d'un appel du client vers le serveur, la session ne contenant alors que le store ;

Les exemples **[nuxt-05]** et **[nuxt-06]** avaient pour but de montrer comment on pouvait assurer la continuité de l'application lorsque l'utilisateur force l'appel au serveur en tapant manuellement des URL. On rappelle que le comportement par défaut dans ce cas est un redémarrage de l'application, son état du moment étant alors perdu.

4.10 Exemple [nuxt-07] : les contextes client et serveur

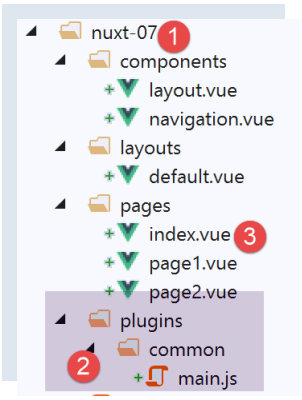
4.10.1 Présentation

L'exemple **[nuxt-07]** vise à explorer l'objet **[context]** du côté serveur et du côté client. Il ne faut pas oublier que ces deux acteurs des applications **[nuxt]** sont séparés : **ils ne partagent** rien sauf :

- ce que le serveur veut bien envoyer au client (dans la réponse HTTP et la page envoyée) ;
- ce que le client veut bien envoyer au serveur (dans sa requête HTTP) ;

Si donc, comme on va le voir, les objets manipulés par le serveur et ceux manipulés par le client **portent le même nom**, ils ne sont pas identiques : ils peuvent parfois être des copies l'un de l'autre mais n'ont jamais la même référence. Modifier un objet client n'a aucun effet sur l'objet de même nom, côté serveur, et vice-versa.

L'exemple **[nuxt-07]** est obtenu initialement par recopie de l'exemple **[nuxt-01]** :



- en **[2]**, nous allons ajouter un plugin commun au client et au serveur ;
- en **[3]**, nous allons modifier quelque peu la page **[index]** ;

4.10.2 Le plugin `[common / main.js]`

Le plugin **[common / main.js]** est exécuté à la fois par le client et le serveur : cela est dû à la configuration **[nuxt.config.js]** suivante :

```
1.  /*
2.     ** Plugins to load before mounting the App
3.     */
4.  plugins: [{ src: '~/plugins/common/main.js' }],
```

- ligne 4 : l'absence de la propriété **[mode]** fait que le plugin **[~/plugins/common/main.js]** sera exécuté à la fois par le client et le serveur : le serveur d'abord, le client ensuite ;

Ce plugin sera le suivant :

```
1.  /* eslint-disable no-undef */
2.  /* eslint-disable no-console */
3.  export default function(...args) {
4.    // qui exécute ce code ?
5.    console.log('[main server], process.server=', process.server, 'process.client=', process.client)
6.    const who = process.server ? 'server' : 'client'
7.    const main = '[main ' + who + ']'
8.    // nbre d'arguments
9.    console.log(main + ', il y a', args.length, 'arguments')
10.
11.   // 1er argument
12.   const context = args[0]
13.   // clés du contexte
14.   dumpkeys(main + ', context', context)
15.   // l'application
16.   dumpkeys(main + ', context.app', context.app)
17.   // la route
18.   dumpkeys(main + ', context.route', context.route)
19.   console.log(main + ', context.route=', context.route)
20.   // le router
21.   dumpkeys(main + ', context.app.router', context.app.router)
22.   // le router.options.routes
23.   dumpkeys(main + ', context.app.router.options.routes', context.app.router.options.routes)
24.   console.log(main + ', context.app.router.options.routes=', context.app.router.options.routes)
25.
26.   // 2ième argument
27.   const inject = args[1]
28.   console.log('inject=', typeof inject)
29. }
30.
31. function dumpkeys(message, object) {
32.   // liste des clés de [object]
33.   const ligne = 'Liste des clés [' + message + ']'
34.   console.log(ligne)
35.   // liste des clés
36.   if (object) {
37.     console.log(Object.keys(object))
38.   }
39. }
```

- lignes 31-39 : la fonction **[dumpkeys]** liste les propriétés de l'objet passé en 2ième paramètre. Cette liste est précédée du message passé en 1er paramètre ;
- ligne 3 : on veut savoir combien d'arguments reçoit la fonction. Pour cela, on utilise la notation **[...args]** qui va avoir pour effet de mettre les paramètres effectifs de la fonction dans le tableau **[args]**. On va découvrir qu'il y a deux arguments ;
- ligne 5 : on affiche qui exécute le code, du serveur ou du client ;
- ligne 6 : l'exécuteur du code, client ou serveur ;
- ligne 7 : une constante chaîne de caractères utilisée dans les logs ;
- ligne 12 : on va découvrir que le 1er argument reçu par la plugin est le contexte de l'exécuteur ;
- ligne 14 : liste des clés de l'objet **[context]** ;
- ligne 16 : on va découvrir que l'objet **[context]** a une propriété **[app]** qui représente l'application **[nuxt]** ;
- ligne 18 : on va découvrir que l'objet **[context]** a une propriété **[route]** qui représente la route courante du routeur ;
- ligne 21 : on va découvrir que l'objet **[app]** a une propriété **[router]** qui représente le routeur ;
- ligne 23 : l'objet **[router.options.routes]** représente les différentes routes de l'application ;
- lignes 27-28 : le second argument du plugin est la fonction **[inject]** que nous avons utilisée dans l'exemple **[nuxt-06]** ;

4.10.3 Le plugin exécuté par le serveur

A l'exécution, le serveur affiche la chose suivante :

```
1.  [main server], process.server= true process.client= false
2.  [main server], il y a 2 arguments
3.  Liste des clés [[main server], context]
4.  [
5.    'isStatic',
6.    'isDev',
7.    'isHMR',
8.    'app',
9.    'payload',
10.   'error',
11.   'base',
12.   'env',
13.   'req',
14.   'res',
15.   'ssrContext',
16.   'redirect',
17.   'beforeNuxtRender',
18.   'route',
19.   'next',
20.   '_redirected',
21.   '_errored',
22.   'params',
23.   'query',
24.   '$axios'
25. ]
```

- lignes 11-12 : nous avons déjà eu l'occasion d'utiliser les propriétés **[base]** et **[env]** dont les valeurs proviennent du fichier **[nuxt.config.js]** ;
- ligne 8 : la propriété **[app]** désigne l'application **[nuxt]** ;
- ligne 18 : la propriété **[route]** désigne la route courante du routeur, ç-à-d la page que va envoyer le serveur ;
- ligne 13 : la requête HTTP du navigateur client ;
- ligne 14 : la réponse HTTP du serveur ;

La liste des propriétés de **[context.app]** est la suivante :

```
1.  Liste des clés [[main server], context.app]
2.  [
3.    'router',
4.    'nuxt',
5.    'head',
6.    'render',
7.    'data',
8.    'beforeCreate',
9.    'created',
10.   'mounted',
11.   'watch',
12.   'computed',
13.   'methods',
14.   'components',
15.   'context',
16.   '$axios'
17. ]
```

- ligne 3 : la propriété **[router]** nous donne accès au router de l'application. C'est important sous **[nuxt]** puisque le routeur est défini par **[nuxt]** lui-même et non par le développeur. Cette propriété est un accès donné au développeur pour modifier le routeur ;

La liste des propriétés de **[context.route]** sont les suivantes :

```
1.  Liste des clés [[main server], context.route]
2.  [
3.    'name',
4.    'meta',
5.    'path',
6.    'hash',
7.    'query',
8.    'params',
9.    'fullPath',
10.   'matched'
11. ]
```

La route **[context.route]** au démarrage du serveur est la suivante :

```
1.  [main server], context.route= {
2.    name: 'index',
3.    meta: [
4.      {}
5.    ],
6.    path: '/',
7.    hash: '',
8.    query: {},
9.    params: {},
10.   fullPath: '/',
11.   matched: [
12.     {
13.       path: '',
14.       regex: /^(?:\/(?=$))?$/i,
15.       components: [Object],
16.       instances: {},
17.       name: 'index',
18.       parent: undefined,
19.       matchAs: undefined,
20.       redirect: undefined,
21.       beforeEnter: undefined,
22.       meta: {},
23.       props: {}
24.     }
25.   ]
26. }
```

- ligne 2 : on voit que la prochaine page du serveur est **[index]** et que son chemin est **[/]** (ligne 10) ;
- ligne 22 : la propriété **[meta]** permet d'ajouter des propriétés aux routes ;

Les propriétés du routeur **[context.app.router]** du serveur sont les suivantes :

```
1.  Liste des clés [[main server], context.app.router]
2.  [
3.    'app',
4.    'apps',
5.    'options',
6.    'beforeHooks',
7.    'resolveHooks',
8.    'afterHooks',
9.    'matcher',
10.   'fallback',
11.   'mode',
12.   'history'
13. ]
```

C'est dans la propriété **[context.app.router.options.routes]** qu'on trouve les différentes routes de l'application :

```
1.  Liste des clés [[main server], context.app.router.options.routes]
2.  [
3.    '0',
4.    '1',
5.    '2'
```

```
6.  ]
7.  [main server], context.app.router.options.routes= [
8.    {
9.      path: '/page1',
10.     component: [Function: _d7b6c762],
11.     name: 'page1'
12.   },
13.   {
14.     path: '/page2',
15.     component: [Function: _d79a9860],
16.     name: 'page2'
17.   },
18.   {
19.     path: '/',
20.     component: [Function: _31eaad9f],
21.     name: 'index'
22.   }
23. ]
```

Enfin le 2ième argument :

```
inject= function
```

4.10.4 La page [index] du serveur

La page **[index]** est la suivante :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">
8.        Home
9.      </b-alert>
10.   </Layout>
11. </template>
12.
13. <script>
14. /* eslint-disable no-undef */
15. /* eslint-disable no-console */
16. /* eslint-disable nuxt/no-env-in-hooks */
17.
18. import Navigation from '@/components/navigation'
19. import Layout from '@/components/layout'
20.
21. export default {
22.   name: 'Home',
23.   // composants utilisés
24.   components: {
25.     Layout,
26.     Navigation
27.   },
28.   data() {
29.     return {
30.       who: process.server ? 'server' : 'client'
31.     }
32.   },
33.
34.   // cycle de vie
35.   beforeCreate() {
36.     // client et serveur
37.     console.log('[home beforeCreate]')
38.   },
39.   created() {
40.     // client et serveur
41.     console.log('[home ' + this.who + ' created]')
42.     this.dumpkeys('[home ' + this.who + ' created], this.$nuxt', this.$nuxt)
43.     this.dumpkeys('[home ' + this.who + ' created], this.$nuxt.context', this.$nuxt.context)
44.   },
45.   beforeMount() {
46.     // client seulement
47.     console.log('[home ' + this.who + ' beforeMount]')
48.   },
49.   mounted() {
```

```
50.     // client seulement
51.     console.log('[home ' + this.who + ' mounted]')
52.   },
53.   methods: {
54.     dumpkeys(message, object) {
55.       // liste des clés de [object]
56.       const ligne = 'Liste des clés [' + message + ']'
57.       console.log(ligne)
58.       if (object) {
59.         console.log(Object.keys(object))
60.       }
61.     }
62.   }
63. }
64. </script>
```

- ligne 43 : on a déjà vu que le contexte d'une page pouvait être trouvé dans **[this.$nuxt.context]** ;
- ligne 42 : on affiche également les propriétés de l'objet **[this.$nuxt]** ;

Exécutée par le serveur, cette page donne naissance aux logs suivants :

```
1.  [home beforeCreate]
2.  [home server created]
3.  Liste des clés [[home server created], this.$nuxt]
4.  [
5.    '_uid',
6.    '_isVue',
7.    '$options',
8.    '_renderProxy',
9.    '_self',
10.   '$parent',
11.   '$root',
12.   '$children',
13.   '$refs',
14.   '_watcher',
15.   '_inactive',
16.   '_directInactive',
17.   '_isMounted',
18.   '_isDestroyed',
19.   '_isBeingDestroyed',
20.   '_events',
21.   '_hasHookEvent',
22.   '_vnode',
23.   '_staticTrees',
24.   '$vnode',
25.   '$slots',
26.   '$scopedSlots',
27.   '_c',
28.   '$createElement',
29.   '$attrs',
30.   '$listeners',
31.   '_routerRoot',
32.   '_router',
33.   '_route',
34.   '_bv__modal',
35.   '_bv__toast',
36.   '_vueMeta',
37.   'nuxt',
38.   '_watchers',
39.   'refreshOnlineStatus',
40.   'refresh',
41.   'errorChanged',
42.   'setLayout',
43.   'loadLayout',
44.   '_data',
45.   'layoutName',
46.   'layout',
47.   'isOnline',
48.   '_computedWatchers',
49.   'isOffline',
50.   'error',
51.   'context'
52. ]
```

Les propriétés du contexte serveur dans la page **[index]** sont les suivantes :

```
1.  Liste des clés [[home server created], this.$nuxt.context]
2.  [
3.    'isStatic',
4.    'isDev',
5.    'isHMR',
6.    'app',
7.    'payload',
8.    'error',
9.    'base',
10.   'env',
11.   'req',
12.   'res',
13.   'ssrContext',
14.   'redirect',
15.   'beforeNuxtRender',
16.   'route',
17.   'next',
18.   '_redirected',
19.   '_errored',
20.   'params',
21.   'query',
22.   '$axios'
23. ]
```

Ce sont les mêmes propriétés que dans l'objet **[context]** du plugin.

4.10.5 Le plugin exécuté par le client

Une fois que le serveur a envoyé la page **[index]** au navigateur client, les scripts client prennent la main. Le plugin **[main.js]** va alors être exécuté. Ces logs sont les suivants :

```
1.  [main server], process.server= false process.client= true
2.  [main client], il y a 2 arguments
3.  Liste des clés [[main client], context]
4.  Array(17)0: "isStatic"1: "isDev"2: "isHMR"3: "app"4: "payload"5: "error"6: "base"7: "env"8: "redirect"9:
    "nuxtState"10: "route"11: "next"12: "_redirected"13: "_errored"14: "params"15: "query"16: "$axios"length:
    17__proto__: Array(0)
5.  Liste des clés [[main client], context.app]
6.  Array(14)0: "router"1: "nuxt"2: "head"3: "render"4: "data"5: "beforeCreate"6: "created"7: "mounted"8:
    "watch"9: "computed"10: "methods"11: "components"12: "context"13: "$axios"length: 14__proto__: Array(0)
7.  Liste des clés [[main client], context.route]
8.  Array(8)0: "name"1: "meta"2: "path"3: "hash"4: "query"5: "params"6: "fullPath"7: "matched"length:
    8__proto__: Array(0)
9.  [main client], context.route= ObjectfullPath: "/"hash: ""matched: [{…}]meta: [{…}]name: "index"params:
    {}path: "/"query: {}__proto__: Object
10. Liste des clés [[main client], context.app.router]
11. Array(10)0: "app"1: "apps"2: "options"3: "beforeHooks"4: "resolveHooks"5: "afterHooks"6: "matcher"7:
    "fallback"8: "mode"9: "history"length: 10__proto__: Array(0)
12. Liste des clés [[main client], context.app.router.options.routes]
13. Array(3)0: "0"1: "1"2: "2"length: 3__proto__: Array(0)
14. [main client], context.app.router.options.routes= Array(3)0: {path: "/page1", name: "page1", component: ƒ}1:
    {path: "/page2", name: "page2", component: ƒ}2: {path: "/", name: "index", component: ƒ}length: 3__proto__:
    Array(0)
15. inject= function
```

On retrouve des propriétés analogues à celles trouvées côté serveur avec certaines propriétés qui ont disparu et d'autres qui sont apparues. Ainsi ligne 4, on ne retrouve pas les propriétés **[req, res]** qui étaient les requêtes HTTP du navigateur client et la réponse HTTP du serveur.

4.10.6 La page [index] du client

La page **[index]** du client produit les logs suivants :

```
1.  [home beforeCreate]
2.  [home client created]
3.  Liste des clés [[home client created], this.$nuxt]
4.  (51) ["_uid", "_isVue", "$options", "_renderProxy", "_self", "$parent", "$root", "$children", "$refs",
    "_watcher", "_inactive", "_directInactive", "_isMounted", "_isDestroyed", "_isBeingDestroyed", "_events",
    "_hasHookEvent", "_vnode", "_staticTrees", "$vnode", "$slots", "$scopedSlots", "_c", "$createElement",
    "$attrs", "$listeners", "_routerRoot", "_router", "_route", "_bv__modal", "_bv__toast", "_vueMeta", "nuxt",
    "_watchers", "refreshOnlineStatus", "refresh", "errorChanged", "setLayout", "loadLayout", "_data",
    "layoutName", "layout", "isOnline", "_computedWatchers", "isOffline", "error", "context", "_name",
    "setTransitions", "$loading", "$el"]
```

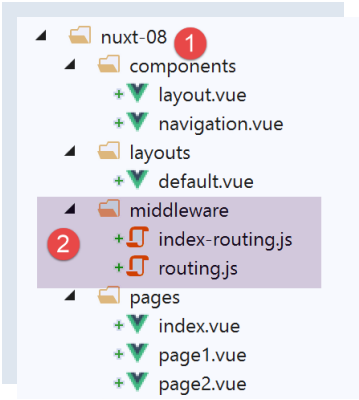
```
5.  Liste des clés [[home client created], this.$nuxt.context]
6.  (17) ["isStatic", "isDev", "isHMR", "app", "payload", "error", "base", "env", "redirect", "nuxtState",
    "route", "next", "_redirected", "_errored", "params", "query", "$axios"]
7.  [home client beforeMount]
8.  [home client mounted]
```

- ligne 4 : les propriétés de l'objet **[this.$nuxt]**. C'est un objet riche avec 51 propriétés ;
- ligne 6 : les propriétés de l'objet **[this.$nuxt.context]**. On retrouve les mêmes propriétés que dans l'objet **[context]** du plugin client ;

4.11 Exemple [nuxt-08] : middlewares de routage

Dans cet exemple, nous introduisons la notion de middlewares de routage, des scripts exécutés à chaque changement de route.

L'exemple **[nuxt-08]** est obtenu initialement par recopie du projet **[nuxt-01]** :



Les middlewares de routage doivent être dans un dossier appelé **[middleware]** [2]. Il peut y avoir un routage à deux niveaux :

- un routage appliqué à chaque navigation. Celui-ci fait alors l'objet d'une déclaration dans le fichier **[nuxt.config.js]** ;
- un routage appliqué à une page particulière, lorsque celle-ci est la cible du routage. Ce routage est alors déclaré dans cette page cible ;

4.11.1 Routage général

Le fichier **[middleware / routing.js]** assurera le routage général. Il fait l'objet de la déclaration suivante dans le fichier **[nuxt.config.js]** :

```
1.    // répertoire du code source
2.    srcDir: 'nuxt-08',
3.    router: {
4.      base: '/nuxt-08/',
5.      middleware: ['routing']
6.    },
7.    // serveur
8.    server: {
9.      port: 81, // default: 3000
10.     host: 'localhost' // default: localhost
11.   }
```

Les middlewares de routage général sont une propriété du routeur (ligne 1). Il peut y avoir plusieurs middlewares de routage. C'est pourquoi ligne 5, la valeur de la propriété **[middleware]** est un tableau. On voit qu'on n'utilise pas de chemin pour désigner le middleware. Il sera automatiquement cherché dans le dossier **[middleware]** du projet ;

Le middleware **[routing]** ne fait ici que des logs :

```
1.  /* eslint-disable no-undef */
2.  /* eslint-disable no-console */
3.  export default function(...args) {
4.    // qui exécute ce code ?
5.    console.log('[routing], process.server=', process.server, 'process.client=', process.client)
6.    const who = process.server ? 'server' : 'client'
7.    const routing = '[routing ' + who + ']'
8.    // nbre d'arguments
9.    console.log(routing + ', il y a', args.length, 'argument(s)')
```

```
10.
11.   // 1er argument
12.   const context = args[0]
13.   // clés du contexte
14.   dumpkeys(routing + ', context', context)
15.   // l'application
16.   dumpkeys(routing + ', context.app', context.app)
17.   // la route
18.   dumpkeys(routing + ', context.route', context.route)
19.   console.log(routing + ', context.route=', context.route)
20.   // le router
21.   dumpkeys(routing + ', context.app.router', context.app.router)
22.   // le router.options.routes
23.   dumpkeys(routing + ', context.app.router.options.routes', context.app.router.options.routes)
24.   console.log(routing + ', context.app.router.options.routes=', context.app.router.options.routes)
25. }
26.
27. function dumpkeys(message, object) {
28.   // liste des clés de [object]
29.   const ligne = 'Liste des clés [' + message + ']'
30.   console.log(ligne)
31.   // liste des clés
32.   if (object) {
33.     console.log(Object.keys(object))
34.   }
35. }
```

- ligne 3 : on va découvrir que le middleware reçoit un argument : le contexte de l'exécuteur (serveur ou client) ;
- lignes 4-25 : on affiche les propriétés de différents objets pour savoir ce qui est utilisable. On va découvrir que le contexte du middleware est quasi identique au contexte du plugin ;

4.11.2 Routage pour une page particulière

On veut contrôler la façon dont on arrive à la page **[index]**. Pour cela, il nous faut introduire la propriété **[middleware]** dans cette page **[index]** :

```
1.  <script>
2.  /* eslint-disable no-undef */
3.  /* eslint-disable no-console */
4.  /* eslint-disable nuxt/no-env-in-hooks */
5.
6.  import Navigation from '@/components/navigation'
7.  import Layout from '@/components/layout'
8.
9.  export default {
10.   name: 'Home',
11.   // composants utilisés
12.   components: {
13.     Layout,
14.     Navigation
15.   },
16.   // cycle de vie
17.   beforeCreate() {
18.     // client et serveur
19.     console.log('[home beforeCreate]')
20.   },
21.   created() {
22.     // client et serveur
23.     console.log('[home created]')
24.   },
25.   beforeMount() {
26.     // client seulement
27.     console.log('[home beforeMount]')
28.   },
29.   mounted() {
30.     // client seulement
31.     console.log('[home mounted]')
32.   },
33.   // routage
34.   middleware: ['index-routing']
35. }
36. </script>
```

- ligne 34 : la propriété **[middleware]** liste les scripts à exécuter à chaque fois que la prochaine page affichée est la page **[index]**. Là encore, ces scripts seront cherchés dans le dossier **[middleware]** du projet ;

Le middleware **[index-routing]** est le suivant :

```
1.  /* eslint-disable no-undef */
2.  /* eslint-disable no-console */
3.  export default function(...args) {
4.    // qui exécute ce code ?
5.    console.log('[index-routing], process.server=', process.server, 'process.client=', process.client)
6.    const who = process.server ? 'server' : 'client'
7.    const indexRouting = '[index-routing ' + who + ']'
8.    // nbre d'arguments
9.    console.log(indexRouting + ', il y a', args.length, 'argument(s)')
10.
11.   // 1er argument
12.   const context = args[0]
13.   // clés du contexte
14.   dumpkeys(indexRouting + ', context', context)
15.   // l'application
16.   dumpkeys(indexRouting + ', context.app', context.app)
17.   // la route
18.   dumpkeys(indexRouting + ', context.route', context.route)
19.   console.log(indexRouting + ', context.route=', context.route)
20.   // le router
21.   dumpkeys(indexRouting + ', context.app.router', context.app.router)
22.   // le router.options.routes
23.   dumpkeys(indexRouting + ', context.app.router.options.routes', context.app.router.options.routes)
24.   console.log(indexRouting + ', context.app.router.options.routes=', context.app.router.options.routes)
25.   // d'où vient-on ?
26.   if (context.from) {
27.     console.log('from=', context.from)
28.   }
29. }
30.
31. function dumpkeys(message, object) {
32.   // liste des clés de [object]
33.   const ligne = 'Liste des clés [' + message + ']'
34.   console.log(ligne)
35.   // liste des clés
36.   if (object) {
37.     console.log(Object.keys(object))
38.   }
39. }
```

Le code de **[index-routing]** est identique à celui de **[routing]** et produit les mêmes résultats. Ce qui nous intéresse c'est de voir quand ces deux middlewares sont exécutés.

4.11.3 Exécution du projet

Nous exécutons le projet. Les logs sont alors les suivants :

C'est le script **[routing]** qui est exécuté en premier par le serveur :

```
1.  [routing], process.server= true process.client= false
2.  [routing server], il y a 1 argument(s)
3.  Liste des clés [[routing server], context]
4.  [ 'isStatic',
5.    'isDev',
6.    'isHMR',
7.    'app',
8.    'payload',
9.    'error',
10.   'base',
11.   'env',
12.   'req',
13.   'res',
14.   'ssrContext',
15.   'redirect',
16.   'beforeNuxtRender',
17.   'route',
18.   'next',
19.   '_redirected',
20.   '_errored',
21.   'params',
22.   'query',
23.   '$axios' ]
```

```
24. Liste des clés [[routing server], context.app]
25. [ 'router',
26.   'nuxt',
27.   'head',
28.   'render',
29.   'data',
30.   'beforeCreate',
31.   'created',
32.   'mounted',
33.   'watch',
34.   'computed',
35.   'methods',
36.   'components',
37.   'context',
38.   '$axios' ]
39. Liste des clés [[routing server], context.route]
40. [ 'name',
41.   'meta',
42.   'path',
43.   'hash',
44.   'query',
45.   'params',
46.   'fullPath',
47.   'matched' ]
48. [routing server], context.route= { name: 'index',
49.   meta: [ {} ],
50.   path: '/',
51.   hash: '',
52.   query: {},
53.   params: {},
54.   fullPath: '/',
55.   matched:
56.    [ { path: '',
57.        regex: /^(?:\/(?=$))?$/i,
58.        components: [Object],
59.        instances: {},
60.        name: 'index',
61.        parent: undefined,
62.        matchAs: undefined,
63.        redirect: undefined,
64.        beforeEnter: undefined,
65.        meta: {},
66.        props: {} } ] }
67. Liste des clés [[routing server], context.app.router]
68. [ 'app',
69.   'apps',
70.   'options',
71.   'beforeHooks',
72.   'resolveHooks',
73.   'afterHooks',
74.   'matcher',
75.   'fallback',
76.   'mode',
77.   'history' ]
78. Liste des clés [[routing server], context.app.router.options.routes]
79. [ '0', '1', '2' ]
80. [routing server], context.app.router.options.routes= [ { path: '/page1',
81.     component: [Function: _61cefe10],
82.     name: 'page1' },
83.   { path: '/page2',
84.     component: [Function: _61dd1591],
85.     name: 'page2' },
86. { path: '/', component: [Function: _00d5e140], name: 'index' } ]
```

On retrouve là ce qu'on avait obtenu avec les plugins.

- ligne 15 : la propriété **[redirect]** est souvent utilisée dans les middlewares : elle permet de changer la cible du routage en cours ;

Puis, parce que la page qui va être affichée est la page **[index]**, le serveur exécute le script **[index-routing]** et affiche les logs suivants :

```
1.  [index-routing], process.server= true process.client= false
2.  [index-routing server], il y a 1 argument(s)
3.  Liste des clés [[index-routing server], context]
4.  [ 'isStatic',
5.    'isDev',
6.    'isHMR',
```

```
7.    'app',
8.    'payload',
9.    'error',
10.   'base',
11.   'env',
12.   'req',
13.   'res',
14.   'ssrContext',
15.   'redirect',
16.   'beforeNuxtRender',
17.   'route',
18.   'next',
19.   '_redirected',
20.   '_errored',
21.   'params',
22.   'query',
23.   '$axios' ]
24. Liste des clés [[index-routing server], context.app]
25. [ 'router',
26.   'nuxt',
27.   'head',
28.   'render',
29.   'data',
30.   'beforeCreate',
31.   'created',
32.   'mounted',
33.   'watch',
34.   'computed',
35.   'methods',
36.   'components',
37.   'context',
38.   '$axios' ]
39. Liste des clés [[index-routing server], context.route]
40. [ 'name',
41.   'meta',
42.   'path',
43.   'hash',
44.   'query',
45.   'params',
46.   'fullPath',
47.   'matched' ]
48. [index-routing server], context.route= { name: 'index',
49.   meta: [ {} ],
50.   path: '/',
51.   hash: '',
52.   query: {},
53.   params: {},
54.   fullPath: '/',
55.   matched:
56.    [ { path: '',
57.        regex: /^(?:\/(?=$))?$/i,
58.        components: [Object],
59.        instances: {},
60.        name: 'index',
61.        parent: undefined,
62.        matchAs: undefined,
63.        redirect: undefined,
64.        beforeEnter: undefined,
65.        meta: {},
66.        props: {} } ] }
67. Liste des clés [[index-routing server], context.app.router]
68. [ 'app',
69.   'apps',
70.   'options',
71.   'beforeHooks',
72.   'resolveHooks',
73.   'afterHooks',
74.   'matcher',
75.   'fallback',
76.   'mode',
77.   'history' ]
78. Liste des clés [[index-routing server], context.app.router.options.routes]
79. [ '0', '1', '2' ]
80. [index-routing server], context.app.router.options.routes= [ { path: '/page1',
81.     component: [Function: _61cefe10],
82.     name: 'page1' },
83.   { path: '/page2',
```

```
84.     component: [Function: _61dd1591],
85.     name: 'page2' },
86. { path: '/', component: [Function: _00d5e140], name: 'index' } ]
```

Les résultats obtenus avec le script **[index-routing]** sont analogues à ceux obtenus avec le script **[routing]**.

Une fois la page **[index]** reçue par le navigateur client, les scripts client prennent la main. Les logs deviennent les suivants :

```
1.  [home beforeCreate]
2.  [home created]
3.  [home beforeMount]
4.  [home mounted]
```

On voit donc qu'au démarrage de l'application **le client n'exécute aucun middleware**. Cela veut dire que cela se produira à chaque fois que l'utilisateur forcera un appel au serveur. Les middlewares ne sont exécutés par le client que lors d'une navigation au sein du client. Naviguons par exemple vers la page **[page1]** (nous sommes sur la page **[index]**) avec le lien **[Page 1]**. Les logs sont alors les suivants :

```
1.  [routing], process.server= false process.client= true
2.  [routing client], il y a 1 argument(s)
3.  Liste des clés [[routing client], context]
4.  (18) ["isStatic", "isDev", "isHMR", "app", "payload", "error", "base", "env", "redirect", "nuxtState",
    "route", "next", "_redirected", "_errored", "params", "query", "$axios", "from"]
5.  Liste des clés [[routing client], context.app]
6.  (14) ["router", "nuxt", "head", "render", "data", "beforeCreate", "created", "mounted", "watch", "computed",
    "methods", "components", "context", "$axios"]
7.  Liste des clés [[routing client], context.route]
8.  (8) ["name", "meta", "path", "hash", "query", "params", "fullPath", "matched"]
9.  [routing client], context.route= {name: "page1", meta: Array(1), path: "/page1", hash: "", query: {…}, …}
10. Liste des clés [[routing client], context.app.router]
11. (10) ["app", "apps", "options", "beforeHooks", "resolveHooks", "afterHooks", "matcher", "fallback", "mode",
    "history"]
12. Liste des clés [[routing client], context.app.router.options.routes]
13. (3) ["0", "1", "2"]
14. [routing client], context.app.router.options.routes= (3) [{…}, {…}, {…}]0: {path: "/page1", name: "page1",
    component: ƒ}1: {path: "/page2", name: "page2", component: ƒ}2: {path: "/", name: "index", component:
    ƒ}length: 3__proto__: Array(0)
15. [page1 beforeCreate]
16. [page1 created]
17. [page1 beforeMount]
18. [page1 mounted]
```

- ligne 2 : le middleware **[routing]** est exécuté par le client ;
- ligne 4 : notez la propriété **[from]** : c'est la route d'où l'on vient ;
- ligne 9 : **[context.route]** est la route où l'on va ;
- lignes 15-18 : affichage de la page **[page1]** ;

Maintenant revenons à la page **[index]** avec le lien **[Home]**. Les logs sont alors les suivants :

```
1.  [routing], process.server= false process.client= true
2.  [routing client], il y a 1 argument(s)
3.  Liste des clés [[routing client], context]
4.  (18) ["isStatic", "isDev", "isHMR", "app", "payload", "error", "base", "env", "redirect", "nuxtState",
    "route", "next", "_redirected", "_errored", "params", "query", "$axios", "from"]
5.  Liste des clés [[routing client], context.app]
6.  (14) ["router", "nuxt", "head", "render", "data", "beforeCreate", "created", "mounted", "watch", "computed",
    "methods", "components", "context", "$axios"]
7.  Liste des clés [[routing client], context.route]
8.  (8) ["name", "meta", "path", "hash", "query", "params", "fullPath", "matched"]
9.  [routing client], context.route= {name: "index", meta: Array(1), path: "/", hash: "", query: {…}, …}
10. Liste des clés [[routing client], context.app.router]
11. (10) ["app", "apps", "options", "beforeHooks", "resolveHooks", "afterHooks", "matcher", "fallback", "mode",
    "history"]
12. Liste des clés [[routing client], context.app.router.options.routes]
13. (3) ["0", "1", "2"]
14. [routing client], context.app.router.options.routes= (3) [{…}, {…}, {…}]0: {path: "/page1", name: "page1",
    component: ƒ}1: {path: "/page2", name: "page2", component: ƒ}2: {path: "/", name: "index", component:
    ƒ}length: 3__proto__: Array(0)
15. [index-routing], process.server= false process.client= true
16. [index-routing client], il y a 1 argument(s)
17. Liste des clés [[index-routing client], context]
18. (18) ["isStatic", "isDev", "isHMR", "app", "payload", "error", "base", "env", "redirect", "nuxtState",
    "route", "next", "_redirected", "_errored", "params", "query", "$axios", "from"]
```

```
19. Liste des clés [[index-routing client], context.app]
20. (14) ["router", "nuxt", "head", "render", "data", "beforeCreate", "created", "mounted", "watch", "computed",
    "methods", "components", "context", "$axios"]
21. Liste des clés [[index-routing client], context.route]
22. (8) ["name", "meta", "path", "hash", "query", "params", "fullPath", "matched"]
23. [index-routing client], context.route= {name: "index", meta: Array(1), path: "/", hash: "", query: {…}, …}
24. Liste des clés [[index-routing client], context.app.router]
25. (10) ["app", "apps", "options", "beforeHooks", "resolveHooks", "afterHooks", "matcher", "fallback", "mode",
    "history"]
26. Liste des clés [[index-routing client], context.app.router.options.routes]
27. (3) ["0", "1", "2"]
28. [index-routing client], context.app.router.options.routes= (3) [{…}, {…}, {…}]0: {path: "/page1", name:
    "page1", component: ƒ}1: {path: "/page2", name: "page2", component: ƒ}2: {path: "/", name: "index",
    component: ƒ}length: 3__proto__: Array(0)
29. from= {name: "page1", meta: Array(1), path: "/page1", hash: "", query: {…}, …}
30. [home beforeCreate]
31. [home created]
32. [home beforeMount]
33. [home mounted]
```

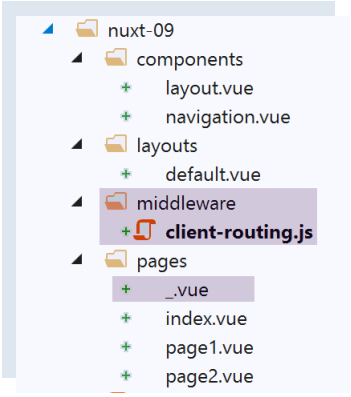
- lignes 1-15 : le client exécute le middleware **[routing]**. C'est normal. Il est exécuté à chaque changement de route;
- lignes 16-29 : le client exécute le middleware **[index-routing]**, ceci parce que :
  - **[index]** est la cible de la route courante (cf ligne 23) ;
  - la page **[index]** a défini un middleware, nommément **[index-routing]** ;

On voit donc que les middlewares de routage général sont exécutés par le client avant les middlewares attachés aux pages.

4.12 Exemple [nuxt-09] : contrôle de la navigation

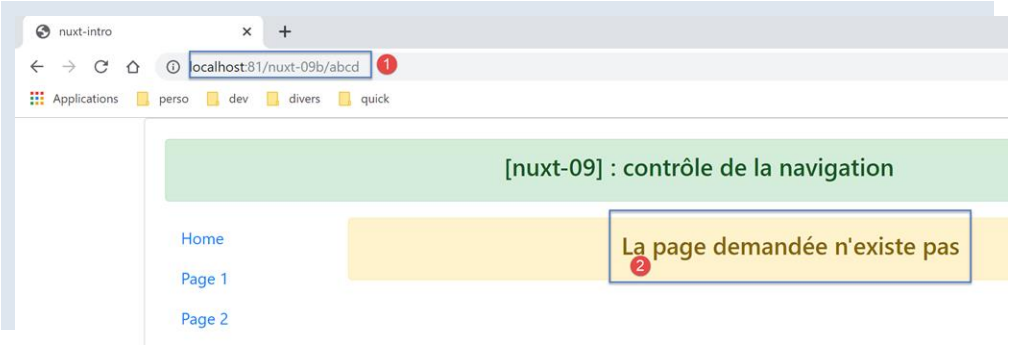
L'exemple **[nuxt-09]** utilise un middleware pour contrôler la navigation du client. Par ailleurs, on ajoute une nouvelle vue pour les cas où l'utilisateur demande au serveur une URL qui n'existe pas dans l'application.

L'exemple **[nuxt-09]** est initialement obtenu par recopie de l'exemple **[nuxt-01]** :



4.12.1 La page [_.vue]

Nous avons dit que les routes de l'application étaient construites à partir du contenu du dossier **[pages]**. Ici, nous avons ajouté dans ce dossier la page **[_.vue]**. Cette page particulière est affichée à chaque fois que l'application est routée vers une page qui n'existe pas. Dans notre exemple ici, cela ne peut pas arriver pour le client. Mais cela peut arriver pour le serveur si par exemple on lui demande l'URL **[/nuxt-09/abcd]**. La page **[abcd]** n'existant pas, c'est la page **[_.vue]** qui va être affichée. Ici, ce sera la suivante :



Le code de la page **[_.vue]** est le suivant :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- mise en page -->
4.    <Layout :left="true" :right="true">
5.      <!-- alerte dans la colonne de droite -->
6.      <template slot="right">
7.        <!-- message sur fond jaune -->
8.        <b-alert show variant="warning" align="center">
9.          <h4>La page demandée n'existe pas</h4>
10.       </b-alert>
11.     </template>
12.     <!-- menu de navigation dans la colonne de gauche -->
13.     <Navigation slot="left" />
14.   </Layout>
15. </template>
16.
17. <script>
18. /* eslint-disable no-undef */
19. /* eslint-disable no-console */
20. /* eslint-disable nuxt/no-env-in-hooks */
21.
22. import Layout from '@/components/layout'
23. import Navigation from '@/components/navigation'
24.
25. export default {
26.   name: '404',
27.   // composants utilisés
28.   components: {
29.     Layout,
30.     Navigation
31.   },
32.   // cycle de vie
33.   beforeCreate() {
34.     // client et serveur
35.     console.log('[404 beforeCreate]')
36.   },
37.   created() {
38.     // client et serveur
39.     console.log('[404 created]')
40.   },
41.   beforeMount() {
42.     // client seulement
43.     console.log('[404 beforeMount]')
44.   },
45.   mounted() {
46.     // client seulement
47.     console.log('[404 mounted]')
48.   }
49. }
50. </script>
```

4.12.2 Le middleware du client

Le code du middleware du client **[client-routing]** est le suivant :

```
1.  /* eslint-disable no-undef */
```

```
2.  /* eslint-disable no-console */
3.  export default function({ route, from, redirect }) {
4.    // seulement le client
5.    if (process.client) {
6.      console.log('[client-routing]')
7.      // contrôle des routes du client
8.      const routes = ['index', 'page1', 'page2', 'index']
9.      // route courante
10.     const current = route.name
11.     // route précédente
12.     const previous = from.name
13.     // on veut une navigation circulaire
14.     // routes[i] vers routes[i+1]
15.     for (let i = 0; i < routes.length - 1; i++) {
16.       if (previous === routes[i] && current !== routes[i + 1]) {
17.         // on reste sur la même page
18.         redirect({ name: routes[i] })
19.         return
20.       }
21.     }
22.   }
23. }
```

- ligne 3 : nous savons que la fonction de routing ne reçoit qu'un paramètre, l'objet **[context]** de celui qui l'exécute, serveur ou client. La notation **[function({ route, from, redirect })]**
  - est équivalente à **[function({ route:route, from:from, redirect:redirect })]** ;
  - ce qui fait que `{ route:route, from:from, redirect:redirect } <-- context` ;
  - ce qui crée trois paramètres **[route, from, redirect]** tels que :
    - route=context.route ;
    - redirect=context.redirect ;
    - from=context.from ;

  La documentation de **[nuxt]** utilise abondamment cette notation. Il faut la connaître ;
- ligne 8 : un tableau des noms de pages dans l'ordre de navigation souhaité
- ligne 10 : le nom de la page de destination du routage courant ;
- lignes 12 : le nom de la page précédente du routage courant ;
- ligne 14 : comme exercice, on ne va autoriser qu'une navigation circulaire **[index --> page1 --> page2 --> index]** ;
- lignes 15-21 : on parcourt le tableau donnant l'ordre de navigation souhaité ;
- ligne 16 : si on découvre que routes**[i]** était la dernière page routée alors la suivante doit être routes**[i+1]** ;
- lignes 18-19 : si ce n'est pas le cas, on redirige l'application vers routes**[i]**, ç-à-d qu'on ne change pas de page : on refuse la navigation ;

4.12.3 Exécution

On exécute l'exemple avec le fichier **[nuxt.config.js]** suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: process.env.npm_package_name || "Introduction à nuxt.js par l'exemple",
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       { hid: 'description', name: 'description', content: process.env.npm_package_description || '' }
12.     ],
13.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
14.   },
15.   /*
16.    ** Customize the progress-bar color
17.    */
18.   loading: { color: '#fff' },
19.   /*
20.    ** Global CSS
21.    */
22.   css: [],
23.   /*
24.    ** Plugins to load before mounting the App
25.    */
26.   plugins: [],
27.   /*
```

```
28.    ** Nuxt.js dev-modules
29.    */
30.   buildModules: [
31.     // Doc: https://github.com/nuxt-community/eslint-module
32.     '@nuxtjs/eslint-module'
33.   ],
34.   /*
35.    ** Nuxt.js modules
36.    */
37.   modules: [
38.     // Doc: https://bootstrap-vue.js.org
39.     'bootstrap-vue/nuxt',
40.     // Doc: https://axios.nuxtjs.org/usage
41.     '@nuxtjs/axios'
42.   ],
43.   /*
44.    ** Axios module configuration
45.    ** See https://axios.nuxtjs.org/options
46.    */
47.   axios: {},
48.   /*
49.    ** Build configuration
50.    */
51.   build: {
52.     /*
53.      ** You can extend webpack config here
54.      */
55.     extend(config, ctx) {}
56.   },
57.   // répertoire du code source
58.   srcDir: 'nuxt-09b',
59.   router: {
60.     base: '/nuxt-09/',
61.     middleware: ['client-routing']
62.   },
63.   // serveur
64.   server: {
65.     port: 81, // default: 3000
66.     host: 'localhost' // default: localhost
67.   }
68. }
```

Vérifiez les points suivants :

- lorsque vous êtes sur la page **[Home]**, vous ne pouvez naviguer que vers la page **[Page 1]** ;
- lorsque vous êtes sur la page **[Page 1]**, vous ne pouvez naviguer que vers la page **[Page 2]** ;
- lorsque vous êtes sur la page **[Page 2]**, vous ne pouvez naviguer que vers la page **[Home]** ;
- lorsque vous demandez une URL incorrecte telle que **[http://localhost:81/nuxt-09/abcd]** alors vous obtenez la vue qui indique que la page demandée n'existe pas ;

4.13 Exemple [nuxt-10] : asyncData et loading

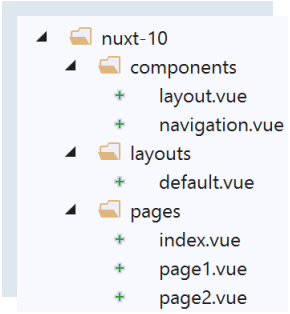
La fonction **[asyncData]** dans une page permet de charger de façon asynchrone des données souvent externes. **[nuxt]** attend la fin de la fonction **[asyncData]** avant de commencer le cycle de vie de la page. Celle-ci n'est donc rendue qu'une fois les données externes obtenues. Elle est exécutée aussi bien par le serveur que par le client avec les règles suivantes :

- lorsque la page est demandée directement au serveur, seul le serveur exécute la fonction **[asyncData]** ;
- ensuite lors de la navigation client, seul le client exécute la fonction **[asyncData]** ;

Au final seul l'un des deux, client ou serveur, exécute la fonction. Par ailleurs, quand c'est le client qui exécute la fonction **[asyncData]**, **[nuxt]** affiche une barre de progression qui peut être paramétrée.

La fonction **[asyncData]** permet de délivrer aux moteurs de recherche des pages avec leurs données ce qui les rend plus signifiantes.

L'exemple **[nuxt-10]** est obtenu initialement par recopie du projet **[nuxt-01]** :



Seule la page **[page1]** évolue.

4.13.1 La page [page1]

Le code de la page **[page1]** est le suivant :

```
<!-- vue n° 1 -->
<template>
  <Layout :left="true" :right="true">
    <!-- navigation -->
    <Navigation slot="left" />
    <!-- message-->
    <b-alert slot="right" show variant="primary"> Page 1 -- result={{ result }} </b-alert>
  </Layout>
</template>

<script>
/* eslint-disable no-console */
/* eslint-disable nuxt/no-timing-in-fetch-data */

import Navigation from '@/components/navigation'
import Layout from '@/components/layout'

export default {
  name: 'Page1',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  // données asynchrones
  asyncData(context) {
    // log
    console.log('[page1 asyncData started]')
    // on rend une promesse
    return new Promise(function(resolve, reject) {
      // on simule une fonction asynchrone
      setTimeout(function() {
        // on rend le résultat asynchrone - un nombre aléatoire ici
        resolve({ result: Math.floor(Math.random() * Math.floor(100)) })
        // log
        console.log('[page1 asyncData finished]')
      }, 5000)
    })
  },

  // cycle de vie
  beforeCreate() {
    console.log('[page1 beforeCreate]')
  },
  created() {
    console.log('[page1 created]')
  },
  beforeMount() {
    console.log('[page1 beforeMount]')
  },
  mounted() {
    console.log('[page1 mounted]')
  }
}
</script>
```

- lignes 26-39 : la fonction **[asyncData]**. Nous avons déjà étudié cette fonction (cf paragraphe lien). Elle est exécutée **avant** le cycle de vie de la page. Pour cette raison, on ne peut utiliser le mot clé **[this]** dans la fonction ;
- ligne 30 : elle doit rendre une promesse **[Promise]** ou utiliser la syntaxe async / await ;
- lignes 32-37 : la fonction asynchrone de la promesse est simulée avec une attente de 5 secondes (ligne 37) ;
- ligne 34 : le résultat de la fonction asynchrone est rendu sous la forme d'un objet {result :...}. L'objet asynchrone rendu par la fonction **[asyncData] est intégré dans l'objet [data] de la page**. C'est pourquoi l'objet **[result]** est-il disponible ligne 7 du template alors même que la page n'avait pas défini d'objet **[data]** ;

4.13.2 Configuration de la barre de progression de [asyncData]

Lorsque la page **[page1]** est la cible d'une navigation au sein du client (mode SPA), le client exécute la fonction **[asyncData]** et **[nuxt]** affiche alors une barre de progression qu'elle cache lorsque la fonction **[asyncData]** a rendu son résultat. La propriété **[loading]** du fichier **[nuxt.config.js]** permet de configurer cette barre :

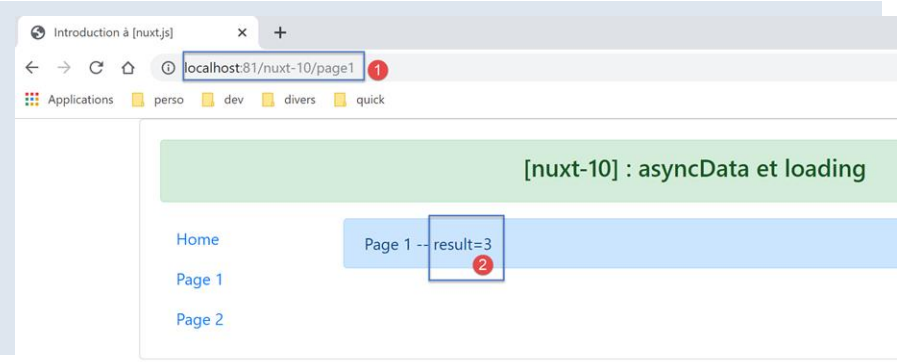
```
1.    /*
2.     ** Customize the progress-bar color
3.     */
4.    loading: {
5.      color: 'blue',
6.      height: '5px',
7.      throttle: 200,
8.      continuous: true
9.    },
```

Par défaut, l'image d'attente de **[nuxt]** est une barre de progression, faisant la largeur de la page. Cette barre a une couleur et une épaisseur. Le trait coloré grandit progressivement de 0 % à 100 % de sa taille, plus ou moins vite donc selon la durée de l'attente.

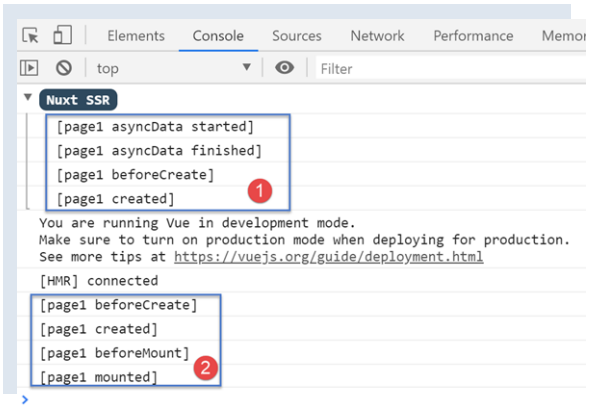
- ligne 5 : fixe la couleur de la barre de progression ;
- ligne 6 : fixe l'épaisseur de la barre en pixels ;
- ligne 7 : **[throttle]** est le délai en millisecondes avant que l'animation ne démarre. Cela permet de ne pas avoir d'image d'animation lorsque la fonction **[asyncData]** rend son résultat rapidement ;
- ligne 8 : **[continuous]** fixe le comportement de l'animation de la barre de progression. Par défaut, la barre grandit progressivement de 0 % à 100 % de sa taille, plus ou moins vite selon la durée de l'attente. Avec **[continuous:true]**, le barre colorée grandit à vitesse constante de 0 à 100 % de sa taille, puis recommence tant que la fonction **[asyncData]** n'a pas rendu son résultat ;

4.13.3 Exécution

Lançons l'application, puis demandons, **à la main**, au serveur la page **[page1]** :



Les logs sont les suivants :



- on voit que seul le serveur **[1]** a exécuté la fonction **[asyncData]** et il l'a fait avant le cycle de la page ;

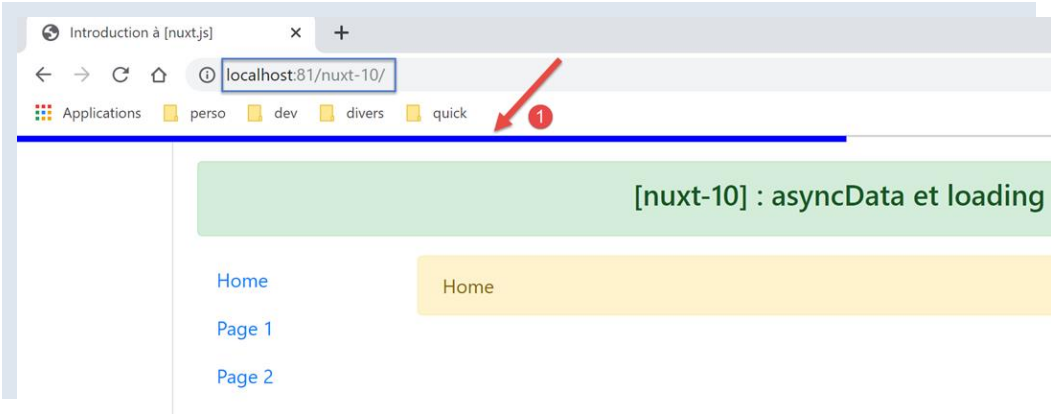
Maintenant examinons la page envoyée par le serveur (code source) :

```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata
    middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-10/">
10.   <link rel="preload" href="/nuxt-10/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-10/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-10/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-10/_nuxt/app.js" as="script">
14. ...
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-
    success">
23.             <h4>[nuxt-10] : asyncData et loading</h4>
24.               </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
29.                     <li class="nav-item">
30.                       <a href="/nuxt-10/" target="_self" class="nav-link">
31.                         Home
32.                       </a>
33.                     </li>
34.                     <li class="nav-item">
35.                       <a href="/nuxt-10/page1" target="_self" class="nav-link active nuxt-link-active">
36.                         Page 1
37.                       </a>
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-10/page2" target="_self" class="nav-link">
41.                         Page 2
42.                       </a>
43.                     </li>
44.                   </ul>
45.                 </div> <div class="col-10"><div role="alert" aria-live="polite" aria-atomic="true"
    class="alert alert-primary"> Page 1 -- result=3 </div></div>
46.               </div>
47.             </div>
48.           </div>
49.         </div>
50.       </div>
51.     </div>
```

```
52.   </div>
53.   <script>window.__NUXT__ = (function (a, b, c) {
54.   return {
55.     layout: "default", data: [{ result: 3 }], error: null, serverRendered: true,
56.     logs: [
57.       { date: new Date(1574939615256), args: ["[page1 asyncData started]"], type: a, level: b, tag: c },
58.       { date: new Date(1574939620263), args: ["[page1 asyncData finished]"], type: a, level: b, tag: c },
59.       { date: new Date(1574939620285), args: ["[page1 beforeCreate]"], type: a, level: b, tag: c },
60.       { date: new Date(1574939620287), args: ["[page1 created]"], type: a, level: b, tag: c }
61.     ]
62.   }
63.     }("log", 2, ""));</script>
64.   <script src="/nuxt-10/_nuxt/runtime.js" defer></script>
65.   <script src="/nuxt-10/_nuxt/commons.app.js" defer></script>
66.   <script src="/nuxt-10/_nuxt/vendors.app.js" defer></script>
67.   <script src="/nuxt-10/_nuxt/app.js" defer></script>
68. </body>
69. </html>
```

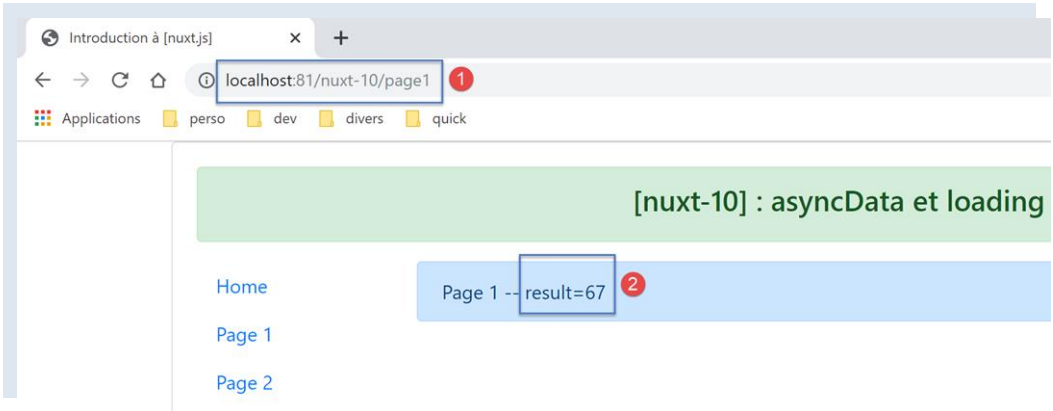
- ligne 55 : on voit que le serveur a envoyé au client un tableau **[data]** qui contient l'objet **[result:3]** qui a été intégré à l'objet **[data]** de la page **[page1]** du serveur. Afin que le client puisse faire de même et donc afficher la même page que le serveur, celui-ci lui transmet l'objet **[result]**. On rappelle que le client ne va pas exécuter la fonction **[asyncData]**. Il va simplement utiliser les données calculées par le serveur ;

Maintenant naviguons de la page **[Home]** à la page **[Page 1]** en utilisant le menu de navigation :

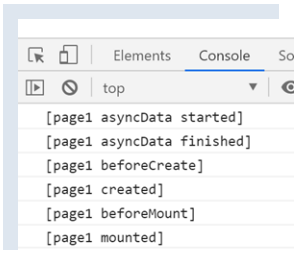


- en **[1]**, on voit apparaître la barre de progression ;

Au bout de 5 secondes, on a la page **[Page 1]** :



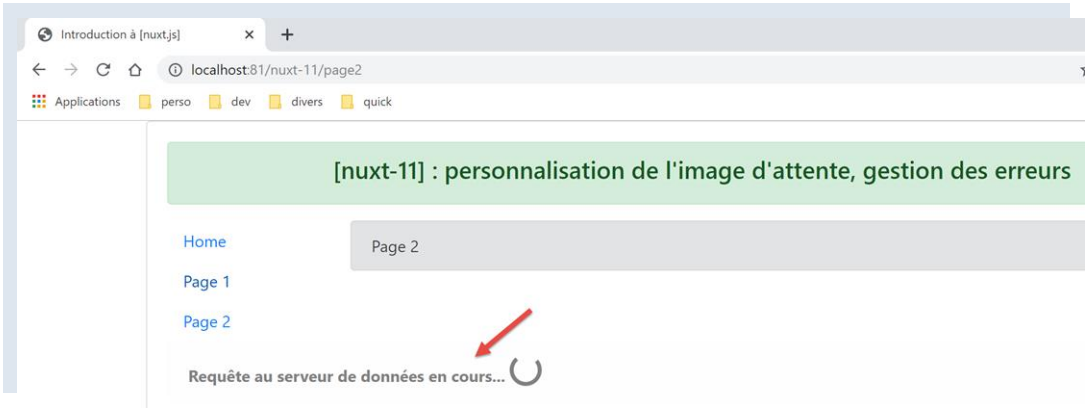
Les logs sont les suivants :



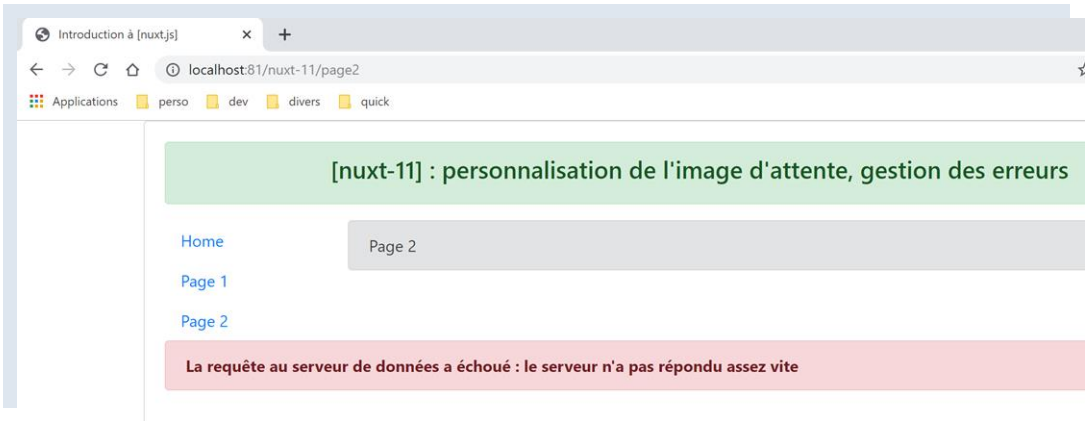
On voit que le client a exécuté la fonction **[asyncData]** avant le cycle de vie de la page.

4.14 Exemple [nuxt-11] : personnalisation de l'image d'attente

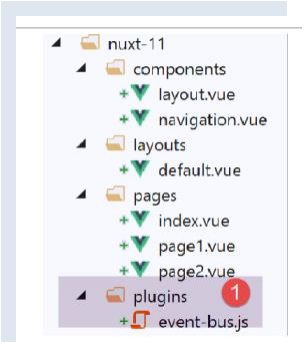
Par défaut, l'image d'attente de **[nuxt]** est une barre de progression. L'exemple **[nuxt-11]** montre qu'on peut la remplacer par sa propre image d'attente :



L'exemple **[nuxt-11]** montre également comment gérer des erreurs de chargement.



L'exemple **[nuxt-11]** est obtenu initialement par recopie de l'exemple **[nuxt-10]** :



Nous allons ajouter en **[1]**, un plugin pour le client dont le rôle sera de gérer des événements entre composants.

4.14.1 Le plugin `[event-bus]`

Le plugin **[event-bus]** sera exécuté par le client et le serveur, mais on verra qu'il ne fonctionne pas côté serveur. Son code est le suivant :

```
1.  /* eslint-disable no-console */
2.  // on crée un bus d'événements entre les vues
3.  import Vue from 'vue'
4.  export default (context, inject) => {
5.    // le bus d'événement
6.    const eventBus = new Vue()
7.    // injection d'une fonction [eventBus] dans le contexte
8.    inject('eventBus', () => eventBus)
9.    // log
10.   console.log('[event-bus créé]')
11. }
```

- ligne 6 : le bus d'événements est une instance de la classe **[Vue]**. En effet, celle-ci a les méthodes pour gérer des événements :
  - **[$emit]** : pour émettre un événement ;
  - **[$on]** : pour se mettre à l'écoute d'un événement particulier ;

  Ce bus d'événements ne gèrera qu'un événement, **[loading]**, qui sera utilisé par les pages pour démarrer / arrêter l'animation d'attente de la fin d'une fonction asynchrone ;
- ligne 8 : on crée une fonction **[$eventBus]** (1er argument) dont le rôle sera de rendre l'objet **[eventBus]** que l'on vient de créer (2ième argument). Cette fonction est injectée dans le contexte pour qu'elle soit disponible dans les objets **[context.app]** et l'objet **[this]** des pages ;

4.14.2 Le layout `[default.vue]`

Le layout **[default.vue]** évolue de la façon suivante :

```
1.  <template>
2.    <div class="container">
3.      <b-card>
4.        <!-- un message -->
5.        <b-alert show variant="success" align="center">
6.          <h4>[nuxt-11] : personnalisation de l'attente, gestion des erreurs</h4>
7.        </b-alert>
8.        <!-- la vue courante du routage -->
9.        <nuxt />
10.       <!-- loading -->
11.       <b-alert v-if="showLoading" show variant="light">
12.         <strong>Requête au serveur de données en cours...</strong>
13.         <div class="spinner-border ml-auto" role="status" aria-hidden="true"></div>
14.       </b-alert>
15.       <!-- erreur de chargement -->
16.       <b-alert v-if="showErrorLoading" show variant="danger">
17.         <strong>La requête au serveur de données a échoué : {{ errorLoadingMessage }}</strong>
18.       </b-alert>
19.     </b-card>
20.   </div>
21. </template>
22.
23. <script>
24. /* eslint-disable no-console */
25. export default {
```

```
26.   name: 'App',
27.   data() {
28.     return {
29.       showLoading: false,
30.       showErrorLoading: false
31.     }
32.   },
33.   // cycle de vie
34.   beforeCreate() {
35.     console.log('[default beforeCreate]')
36.   },
37.   created() {
38.     console.log('[default created]')
39.     // on écoute l'évt [loading]
40.     this.$eventBus().$on('loading', this.mShowLoading)
41.     // ainsi que l'évt [errorLoadingMessage]
42.     this.$eventBus().$on('errorLoading', this.mShowErrorLoading)
43.   },
44.   beforeMount() {
45.     console.log('[default beforeMount]')
46.   },
47.   mounted() {
48.     console.log('[default mounted]')
49.   },
50.   methods: {
51.     // gestion du chargement
52.     mShowLoading(value) {
53.       console.log('[default mShowLoading], showLoading=', value)
54.       this.showLoading = value
55.     },
56.     // erreur de chargement
57.     mShowErrorLoading(value, errorLoadingMessage) {
58.       console.log('[default mShowErrorLoading], showErrorLoading=', value, 'errorLoadingMessage=', errorLoa
    dingMessage)
59.       this.showErrorLoading = value
60.       this.errorLoadingMessage = errorLoadingMessage
61.     }
62.   }
63. }
64. </script>
```

- lignes 11-14 : l'animation d'attente. Elle n'est affichée que si la propriété **[showLoading]** est vraie (ligne 29) ;
- lignes 16-18 : le message d'erreur du chargement. Il n'est affiché que si la propriété **[showErrorLoading]** (ligne 30) est vraie ;
- lignes 29-30 : au chargement initial du composant, l'animation d'attente est cachée ainsi que le message d'erreur ;
- lignes 37-43 : lorsqu'elle est créée, la page écoute l'événement **[loading]** (1er argument) sur le bus d'événements créé par le plugin. A sa réception, elle fait exécuter la méthode **[mShowLoading]** des lignes 52-55 (2ième argument) ;
- lignes 52-55 : la valeur reçue par la méthode **[mShowLoading]** sera un booléen true / false. Elle sert à montrer / cacher le message d'attente ;
- lignes 41-42 : lorsqu'elle est créée, la page écoute l'événement **[errorLoading]** (1er argument) sur le bus d'événements créé par le plugin. A sa réception, elle fait exécuter la méthode **[mShowErrorLoading]** des lignes 57-61 (2ième argument) ;
- ligne 57 : la méthode **[mShowErrorLoading]** reçoit deux arguments :
- le 1er argument est un booléen true / false pour montrer / cacher le message d'erreur ;
- le 2ième argument n'est présent que s'il y a eu erreur. Il représente le message d'erreur à afficher ;
- les logs des lignes 53 et 58 vont nous montrer que les méthodes **[showLoading]** et **[showErrorLoading]** **ne sont pas exécutées** côté serveur ;

4.14.3 La page [page1]

Le code de la page **[page1]** évolue de la façon suivante :

```
1.  <!-- vue n° 1 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="primary"> Page 1 -- result={{ result }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
```

```
13. /* eslint-disable nuxt/no-timing-in-fetch-data */
14.
15. import Navigation from '@/components/navigation'
16. import Layout from '@/components/layout'
17.
18. export default {
19.   name: 'Page1',
20.   // composants utilisés
21.   components: {
22.     Layout,
23.     Navigation
24.   },
25.   // données asynchrones
26.   asyncData(context) {
27.     // log
28.     console.log('[page1 asyncData started]')
29.     // début attente
30.     context.app.$eventBus().$emit('loading', true)
31.     // pas d'erreur
32.     context.app.$eventBus().$emit('errorLoading', false)
33.     // on rend une promesse
34.     return new Promise(function(resolve, reject) {
35.       // on simule une fonction asynchrone
36.       setTimeout(function() {
37.         // fin attente
38.         context.app.$eventBus().$emit('loading', false)
39.         // log
40.         console.log('[page1 asyncData finished]')
41.         // on rend le résultat asynchrone - un nombre aléatoire ici
42.         resolve({ result: Math.floor(Math.random() * Math.floor(100)) })
43.       }, 5000)
44.     })
45.   },
46.
47.   // cycle de vie
48.   beforeCreate() {
49.     console.log('[page1 beforeCreate]')
50.   },
51.   created() {
52.     console.log('[page1 created]')
53.   },
54.   beforeMount() {
55.     console.log('[page1 beforeMount]')
56.   },
57.   mounted() {
58.     console.log('[page1 mounted]')
59.   }
60. }
61. </script>
```

- les modifications ont lieu dans la fonction **[asyncData]** des lignes 26-47 ;
- lignes 29-30 : avant que ne commence la fonction asynchrone on émet l'événement **[loading]** à destination des autres pages de l'application. On rappelle que dans **[asyncData]**, on n'a pas accès à l'objet **[this]** pas encore créé. On utilise alors le contexte que reçoit en argument la fonction **[asyncData]** (ligne 26) ;
- ligne 30 : on utilise le bus d'événements pour indiquer que le chargement va commencer ;
- ligne 38 : on utilise le bus d'événements pour indiquer que le chargement est terminé ;

Note : A l'exécution, lorsque la page **[page1]** est demandée au serveur, on ne voit pas l'image d'attente. Dans les logs on voit que côté serveur la méthode **[default.mShowLoading]** n'est pas appelée. De toute façon, voir l'image d'attente n'a pas de sens lorsque la page est demandée au serveur. Celui-ci n'envoie la page au navigateur client qu'une fois la fonction **[asyncData]** terminée. L'image d'attente est alors inutile. Ce sera le cas pour toutes les pages de l'application demandées directement au serveur.

4.14.4 La page [index]

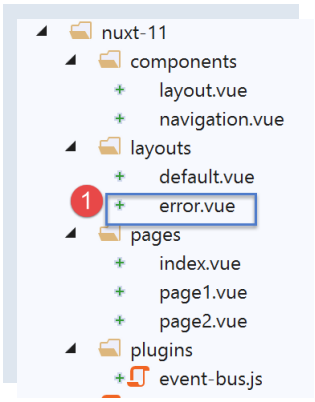
Le code de la page **[index]** est le suivant :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">
8.        Home
```

```
9.      </b-alert>
10.   </Layout>
11. </template>
12.
13. <script>
14. /* eslint-disable no-undef */
15. /* eslint-disable no-console */
16. /* eslint-disable nuxt/no-env-in-hooks */
17. /* eslint-disable nuxt/no-timing-in-fetch-data */
18.
19. import Navigation from '@/components/navigation'
20. import Layout from '@/components/layout'
21.
22. export default {
23.   name: 'Home',
24.   // composants utilisés
25.   components: {
26.     Layout,
27.     Navigation
28.   },
29.   // données asynchrones
30.   asyncData(context) {
31.     // log
32.     console.log('[page1 asyncData started]')
33.     // début attente
34.     context.app.$eventBus().$emit('loading', true)
35.     // pas d'erreur
36.     context.app.$eventBus().$emit('errorLoading', false)
37.     // on rend une promesse
38.     return new Promise(function(resolve, reject) {
39.       // on simule une fonction asynchrone
40.       setTimeout(function() {
41.         // fin attente
42.         context.app.$eventBus().$emit('loading', false)
43.         // log
44.         console.log('[page1 asyncData finished]')
45.         // on rend une erreur
46.         reject(new Error("le serveur n'a pas répondu assez vite"))
47.       }, 5000)
48.     }).catch((e) => context.error({ statusCode: 500, message: e.message }))
49.   },
50.   // cycle de vie
51.   beforeCreate() {
52.     console.log('[home beforeCreate]')
53.   },
54.   created() {
55.     console.log('[home created]')
56.   },
57.   beforeMount() {
58.     console.log('[home beforeMount]')
59.   },
60.   mounted() {
61.     console.log('[home mounted]')
62.     // pas d'erreur
63.     this.$eventBus().$emit('errorLoading', false)
64.   }
65. }
66. </script>
```

- lignes 30-49 : la fonction **[asyncData]** est identique à celle de la page **[page1]** à un détail près : ligne 46, on termine la fonction asynchrone sur un échec (utilisation de la méthode **[reject]**) ;
- ligne 46 : le paramètre de la fonction **[reject]** est une instance de la classe **[Error]**. Le paramètre du constructeur **[Error]** est le message de l'erreur ;
- ligne 48 : cette erreur est interceptée par la méthode **[catch]** de la **[Promise]** qui reçoit l'erreur en paramètre. On utilise alors la fonction **[context.error]** pour déclarer l'erreur. Le paramètre de la fonction **[context.error]** est un objet avec ici deux propriétés :
- **[statusCode]** : un code HTTP d'erreur ;
- **[message]** : un message d'erreur ;

Que **[asyncData]** soit exécutée par le client ou le serveur, en cas d'erreur **[context.error]**, **[nuxt]** affiche la page **[layouts / error.vue]** :



Bien que ce soit une page, la page **[error.vue]** est cherchée dans le dossier **[layouts]** (peut-être pour éviter qu'elle soit incluse dans les routes de l'application ?). Ici, la page **[error.vue]** est la suivante :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- mise en page -->
4.    <Layout :left="true" :right="true">
5.      <!-- alerte dans la colonne de droite -->
6.      <template slot="right">
7.        <!-- message sur fond jaune -->
8.        <b-alert show variant="danger" align="center">
9.          <h4>L'erreur suivante s'est produite : {{ JSON.stringify(error) }}</h4>
10.       </b-alert>
11.     </template>
12.     <!-- menu de navigation dans la colonne de gauche -->
13.     <Navigation slot="left" />
14.   </Layout>
15. </template>
16.
17. <script>
18. /* eslint-disable no-undef */
19. /* eslint-disable no-console */
20. /* eslint-disable nuxt/no-env-in-hooks */
21.
22. import Layout from '@/components/layout'
23. import Navigation from '@/components/navigation'
24.
25. export default {
26.   name: 'Error',
27.   // composants utilisés
28.   components: {
29.     Layout,
30.     Navigation
31.   },
32.   // propriété [props]
33.   props: { error: { type: Object, default: () => 'waiting ...' } },
34.   // cycle de vie
35.   beforeCreate() {
36.     // client et serveur
37.     console.log('[error beforeCreate]')
38.   },
39.   created() {
40.     // client et serveur
41.     console.log('[error created, error=]', this.error)
42.   },
43.   beforeMount() {
44.     // client seulement
45.     console.log('[error beforeMount]')
46.   },
47.   mounted() {
48.     // client seulement
49.     console.log('[error mounted]')
50.   }
51. }
52. </script>
```

Lorsque **[nuxt]** affiche la page **[error.vue]**, elle lui passe en propriété **[props]**, l'erreur qui s'est produite (ligne 33). Si l'erreur a été provoquée par **[context.error(objet1)]**, la propriété **[props]** de la page **[error.vue]** aura la valeur **[objet1]**. La documentation

[nuxt] indique que **[objet1]** doit avoir au moins les attributs **[statusCode, message]**. La ligne 9 affiche la chaîne jSON de l'objet **[objet1]** reçu.

4.14.5 La page [page2]

La page **[page2]** montre une autre façon de gérer l'erreur :

- dans **[page1]**, l'erreur est affichée dans une page à part **[error.vue]** ;
- dans **[page2]**, l'erreur sera affichée dans la page **[page2]** qui a provoqué l'erreur ;

Le code de **[page2]** est le suivant :

```
1.  <!-- vue n° 2 -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-alert slot="right" show variant="secondary">
8.        Page 2
9.      </b-alert>
10.   </Layout>
11. </template>
12.
13. <script>
14. /* eslint-disable no-console */
15. /* eslint-disable nuxt/no-timing-in-fetch-data */
16.
17. import Navigation from '@/components/navigation'
18. import Layout from '@/components/layout'
19.
20. export default {
21.   name: 'Page2',
22.   // composants utilisés
23.   components: {
24.     Layout,
25.     Navigation
26.   },
27.   // données asynchrones
28.   asyncData(context) {
29.     // log
30.     console.log('[page2 asyncData started]')
31.     // début attente
32.     context.app.$eventBus().$emit('loading', true)
33.     // pas d'erreur
34.     context.app.$eventBus().$emit('errorLoading', false)
35.     // on rend une promesse
36.     return new Promise(function(resolve, reject) {
37.       // on simule une fonction asynchrone
38.       setTimeout(function() {
39.         // fin attente
40.         context.app.$eventBus().$emit('loading', false)
41.         // on génére arbitrairement une erreur
42.         const errorLoadingMessage = "le serveur n'a pas répondu assez vite"
43.         // fin avec succès
44.         resolve({ showErrorLoading: true, errorLoadingMessage })
45.         // log
46.         console.log('[page2 asyncData finished]')
47.       }, 5000)
48.     })
49.   },
50.   // cycle de vie
51.   beforeCreate() {
52.     console.log('[page2 beforeCreate]')
53.   },
54.   created() {
55.     console.log('[page2 created]')
56.   },
57.   beforeMount() {
58.     console.log('[page2 beforeMount]')
59.   },
60.   mounted() {
61.     console.log('[page2 mounted]')
62.     // client
63.     if (this.showErrorLoading) {
```

```
64.       console.log('[page2 mounted, showErrorLoading=true]')
65.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
66.     }
67.   }
68. }
69. </script>
```

De nouveau, on insère une fonction **[asyncData]** dans le code de la page et comme **[index]**, **[page2]** va générer une erreur qu'on va cette fois gérer différemment.

- ligne 44 : le serveur comme le client terminent la promesse sur un succès en rendant le résultat **[{ showErrorLoading: true, errorLoadingMessage }]**. On sait que cela va avoir pour effet d'inclure les propriétés **[showerrorLoading, errorLoadingMessage]** dans les propriétés **[data]** de la page et que le client va recevoir ces propriétés ;
- lignes 60-67 : on sait que la fonction **[mounted]** n'est exécutée que par le client ;
- ligne 63 : le client teste si la propriété **[showErrorLoading]** a été positionnée (par le serveur ou le client selon les cas). Si oui, il émet l'événement **['errorLoading']** (ligne 65) pour que la page **[default]** affiche le message d'erreur **[this.errorLoadingMessage]**. Au final, le serveur envoie une page **sans message d'erreur affiché**. Celui-ci est affiché au dernier moment par le client lorsque la page est 'montée' ;

4.14.6 Exécution

4.14.6.1 [nuxt.config]

Le fichier **[nuxt.config.js]** d'exécution est le suivant :

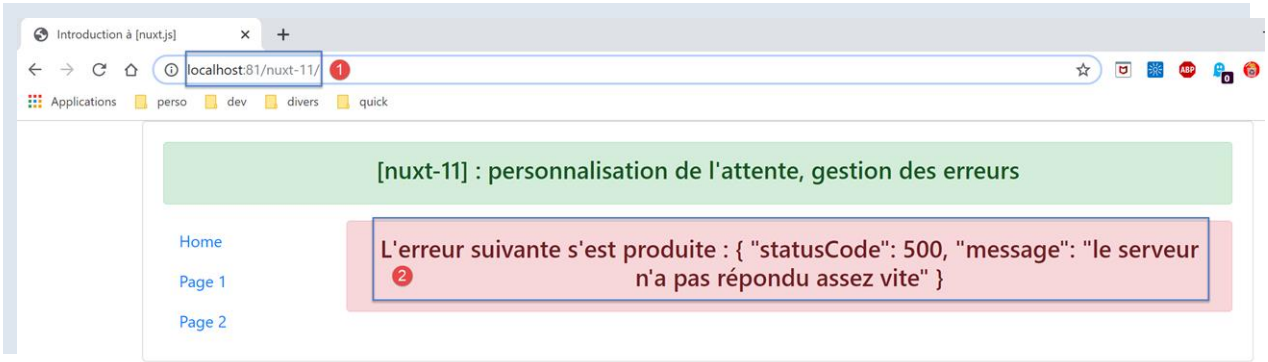
```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: false,
23.
24.   /*
25.    ** Global CSS
26.    */
27.   css: [],
28.   /*
29.    ** Plugins to load before mounting the App
30.    */
31.   plugins: [{ src: '@/plugins/event-bus' }],
32.   /*
33.    ** Nuxt.js dev-modules
34.    */
35.   buildModules: [
36.     // Doc: https://github.com/nuxt-community/eslint-module
37.     '@nuxtjs/eslint-module'
38.   ],
39.   /*
40.    ** Nuxt.js modules
41.    */
42.   modules: [
43.     // Doc: https://bootstrap-vue.js.org
44.     'bootstrap-vue/nuxt',
45.     // Doc: https://axios.nuxtjs.org/usage
46.     '@nuxtjs/axios'
47.   ],
48.   /*
49.    ** Axios module configuration
```

```
50.     ** See https://axios.nuxtjs.org/options
51.     */
52.   axios: {},
53.   /*
54.     ** Build configuration
55.     */
56.   build: {
57.     /*
58.       ** You can extend webpack config here
59.       */
60.     extend(config, ctx) {}
61.   },
62.   // répertoire du code source
63.   srcDir: 'nuxt-11',
64.   // routeur
65.   router: {
66.     // racine des URL de l'application
67.     base: '/nuxt-11/'
68.   },
69.   // serveur
70.   server: {
71.     // port de service, 3000 par défaut
72.     port: 81,
73.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
74.     // 0.0.0.0 = toutes les adresses réseau de la machine
75.     host: 'localhost'
76.   }
77. }
```

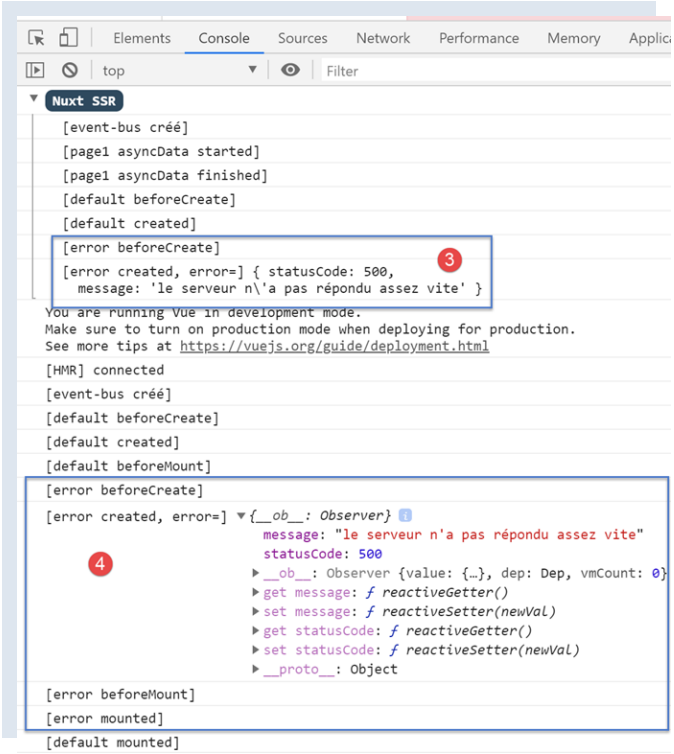
- ligne 22 : on met la propriété **[loading]** à **[false]** pour que **[nuxt]** n'utilise pas son image d'attente par défaut ;
- ligne 31 : le plugin qui définit le bus d'événements ;

4.14.6.2 La page [index] exécutée par le serveur

Demandons la page **[index]** au serveur (on tape l'URL **[http://localhost:81/nuxt-11/]** à la main). La page affichée par le navigateur client est la suivante :



Les logs sont les suivants :



- en **[3]**, on voit que le serveur envoie la page **[error.vue]** ;
- en **[4]**, on voit que le client affiche lui aussi la page **[error]** avec la même erreur que le serveur ;
- on peut remarquer que la méthode **[mShowLoading]** de la page **[default]** n'a pas été appelée côté serveur alors même que la page **[index]** avait activé une attente. Cette méthode est appelée à réception d'un événement et visiblement la gestion événementielle n'est pas implémentée côté serveur ;

Examinons le code source de la page reçue par le navigateur client :

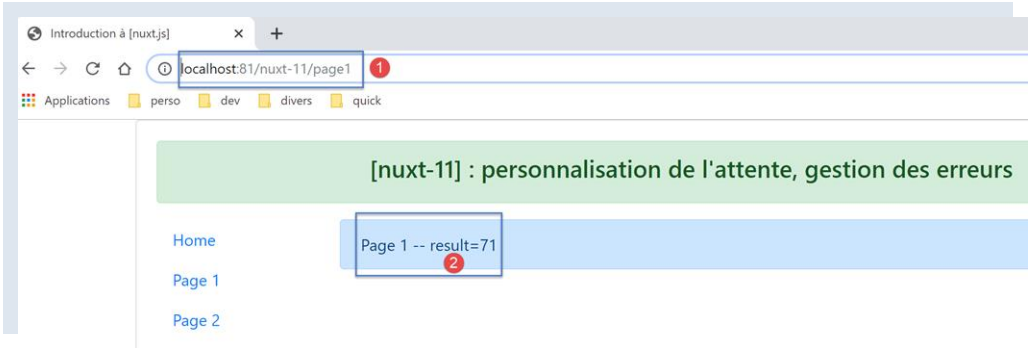
```
1.  <!doctype html>
2.  <html data-n-head-ssr>
3.  <head>
4.    <title>Introduction à [nuxt.js]</title>
5.    <meta data-n-head="ssr" charset="utf-8">
6.    <meta data-n-head="ssr" name="viewport" content="width=device-width, initial-scale=1">
7.    <meta data-n-head="ssr" data-hid="description" name="description" content="ssr routing loading asyncdata
    middleware plugins store">
8.    <link data-n-head="ssr" rel="icon" type="image/x-icon" href="/favicon.ico">
9.    <base href="/nuxt-11/">
10.   <link rel="preload" href="/nuxt-11/_nuxt/runtime.js" as="script">
11.   <link rel="preload" href="/nuxt-11/_nuxt/commons.app.js" as="script">
12.   <link rel="preload" href="/nuxt-11/_nuxt/vendors.app.js" as="script">
13.   <link rel="preload" href="/nuxt-11/_nuxt/app.js" as="script">
14.   ...
15. </head>
16. <body>
17.   <div data-server-rendered="true" id="__nuxt">
18.     <div id="__layout">
19.       <div class="container">
20.         <div class="card">
21.           <div class="card-body">
22.             <div role="alert" aria-live="polite" aria-atomic="true" align="center" class="alert alert-
    success">
23.             <h4>[nuxt-11] : personnalisation de l'attente, gestion des erreurs</h4>
24.               </div>
25.             <div>
26.               <div class="row">
27.                 <div class="col-2">
28.                   <ul class="nav flex-column">
29.                     <li class="nav-item">
30.                       <a href="/nuxt-11/" target="_self" class="nav-link active nuxt-link-active">
31.                         Home
32.                       </a>
33.                     </li>
```

```
34.                     <li class="nav-item">
35.                       <a href="/nuxt-11/page1" target="_self" class="nav-link">
36.                         Page 1
37.                       </a>
38.                     </li>
39.                     <li class="nav-item">
40.                       <a href="/nuxt-11/page2" target="_self" class="nav-link">
41.                         Page 2
42.                       </a>
43.                     </li>
44.                   </ul>
45.                 </div> <div class="col-10"><div role="alert" aria-live="polite" aria-atomic="true"
    align="center" class="alert alert-danger">
46.                 <h4>L'erreur suivante s'est produite :
    {&amp;quot;statusCode&amp;quot;:500,&amp;quot;message&amp;quot;:&amp;quot;le serveur n'a pas répondu assez vite&amp;quot;}</h4>
47.                     </div>
48.                 </div>
49.               </div>
50.             </div>
51.           </div>
52.         </div>
53.       </div>
54.     </div>
55.   </div>
56.   <script>window.__NUXT__ = (function (a, b, c, d) {
57.   d.statusCode = 500; d.message = "le serveur n'a pas répondu assez vite";
58.   return {
59.     layout: "default", data: [d], error: d, serverRendered: true,
60.     logs: [
61.       { date: new Date(1575047424168), args: ["[event-bus créé]"], type: a, level: b, tag: c },
62.       { date: new Date(1575047424175), args: ["[page1 asyncData started]"], type: a, level: b, tag: c },
63.       { date: new Date(1575047429455), args: ["[page1 asyncData finished]"], type: a, level: b, tag: c },
64.       { date: new Date(1575047429515), args: ["[default beforeCreate]"], type: a, level: b, tag: c },
65.       { date: new Date(1575047429675), args: ["[default created]"], type: a, level: b, tag: c },
66.       { date: new Date(1575047430157), args: ["[error beforeCreate]"], type: a, level: b, tag: c },
67.       { date: new Date(1575047430246), args: ["[error created, error=]", "{ statusCode: 500,\n  message:
    'le serveur n\\'a pas répondu assez vite' }"], type: a, level: b, tag: c }]
68.   }
69.     }("log", 2, "", {}));</script>
70.   <script src="/nuxt-11/_nuxt/runtime.js" defer></script>
71.   <script src="/nuxt-11/_nuxt/commons.app.js" defer></script>
72.   <script src="/nuxt-11/_nuxt/vendors.app.js" defer></script>
73.   <script src="/nuxt-11/_nuxt/app.js" defer></script>
74. </body>
75. </html>
```

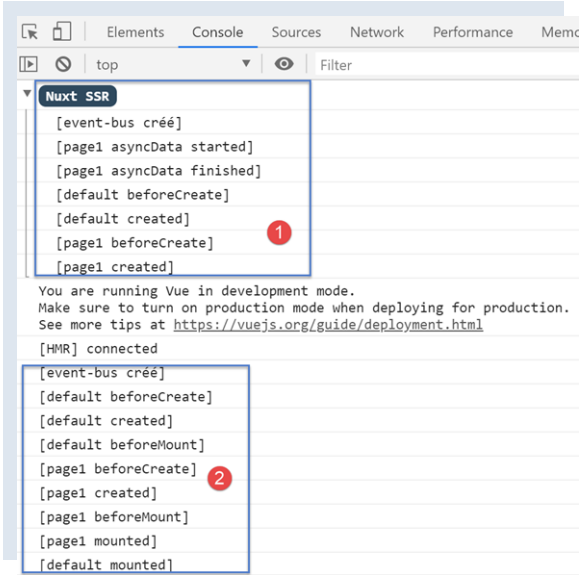
- ligne 57 : on voit que le serveur a envoyé un objet **[d]** qui représente l'erreur qui s'est produite côté serveur ;
- ligne 59 : on voit une propriété **[error]** ayant pour valeur l'objet **[d]**. On peut imaginer que c'est la présence de la propriété **[error]** dans la page envoyée par le serveur qui fait que les scripts client vont afficher la page **[error.vue]** avec l'erreur **[error]** ;

4.14.6.3 La page [page1] exécutée par le serveur

On tape à la main l'URL **[http://localhost:81/nuxt-11/page1]**. Au bout de 5 secondes, le navigateur affiche la page suivante :



Les logs affichés sont les suivants :

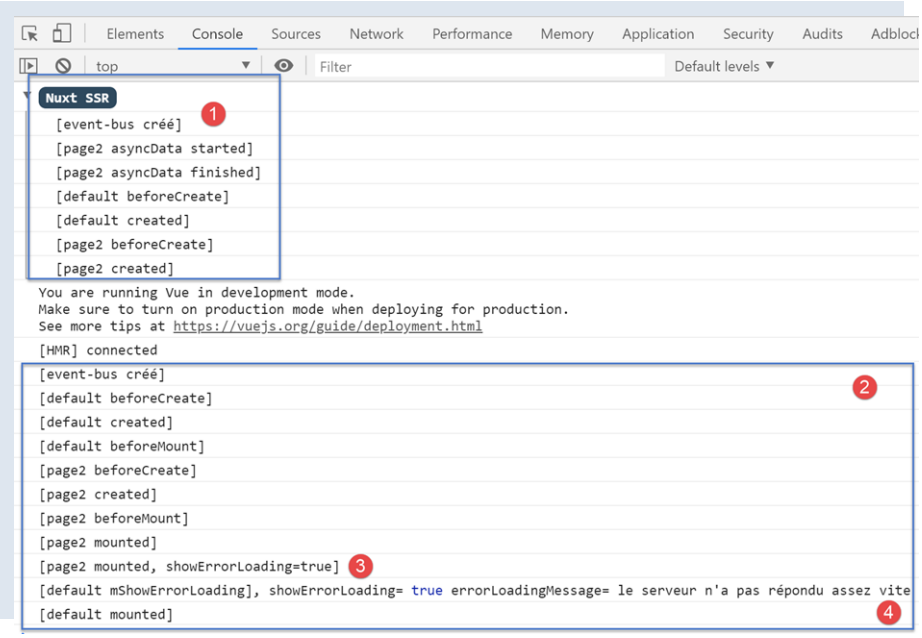


- en **[1]**, les logs du serveur. On peut remarquer que la méthode **[mShowLoading]** de la page **[default]** n'a pas été appelée ;
- en **[2]**, les logs du client ;

4.14.6.4 La page [page2] exécutée par le serveur

Nous tapons à la main l'URL **[http://localhost:81/nuxt-11/page2]**. Au bout de 5 secondes, le navigateur affiche la page suivante :

Examinons les logs affichés dans le navigateur :

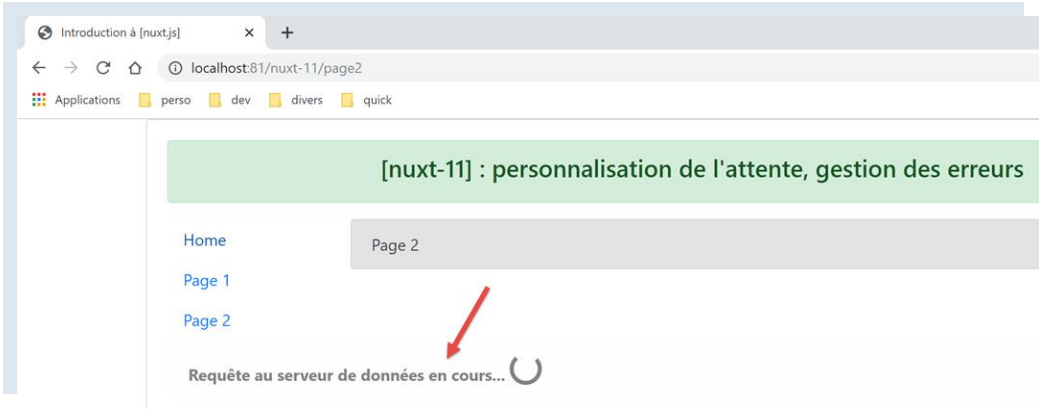


- en **[1]**, les logs du serveur. On rappelle que le serveur a mis les propriétés **[showErrorLoading, errorLoadingMessage]** dans la page envoyée au navigateur client. On sait qu'alors ces propriétés vont être intégrées dans les **[data]** de la page affichée par le client
- en **[3]**, lorsque la page **[page2]** est montée, elle trouve la propriété **[showErrorLoading]** à vrai. Elle envoie alors un événement à la page **[default]**, pour qu'elle affiche le message d'erreur envoyé par le serveur **[4]** ;

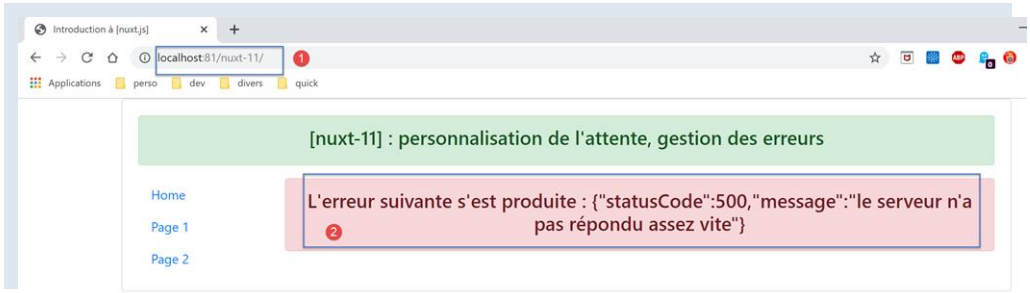
4.14.6.5 La page [index] exécutée par le client

On utilise maintenant les liens de navigation pour afficher les trois pages. Toutes les pages affichées par le client sont identiques à celles affichées par le serveur. La seule différence est que l'image d'attente de la fin des 5 secondes est affichée à chaque fois.

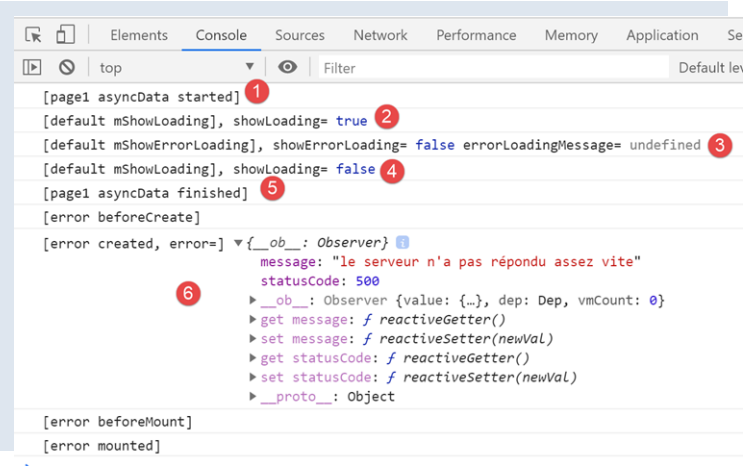
On commence par la page **[index]**. L'image d'attente est alors affichée :



puis au bout de 5 secondes, on obtient la page suivante :



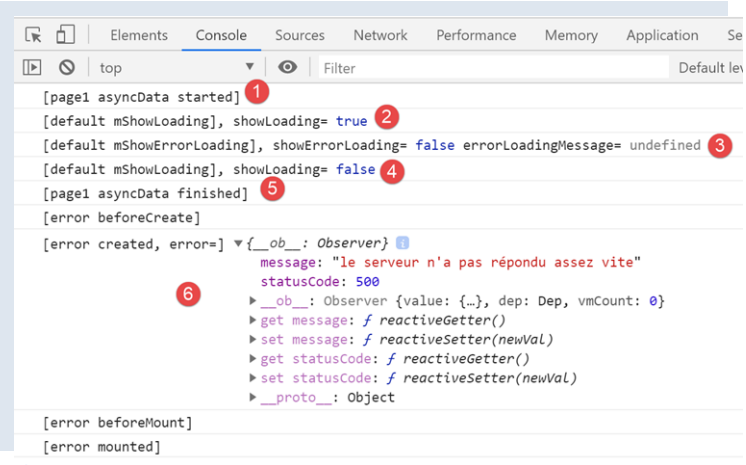
La page finale est donc identique à celle obtenue côté serveur.



Rappelons la fonction **[asyncData]** de la page **[index]** :

```
1.    // données asynchrones
2.    asyncData(context) {
3.      // log
4.      console.log('[page1 asyncData started]')
5.      // début attente
6.      context.app.$eventBus().$emit('loading', true)
7.      // pas d'erreur
8.      context.app.$eventBus().$emit('errorLoading', false)
9.      // on rend une promesse
10.     return new Promise(function(resolve, reject) {
11.       // on simule une fonction asynchrone
12.       setTimeout(function() {
13.         // fin attente
14.         context.app.$eventBus().$emit('loading', false)
15.         // log
16.         console.log('[page1 asyncData finished]')
17.         // on rend une erreur
18.         reject(new Error("le serveur n'a pas répondu assez vite"))
19.       }, 5000)
20.     }).catch((e) => context.error({ statusCode: 500, message: e.message }))
21.   },
```

Les logs du client sont les suivants :

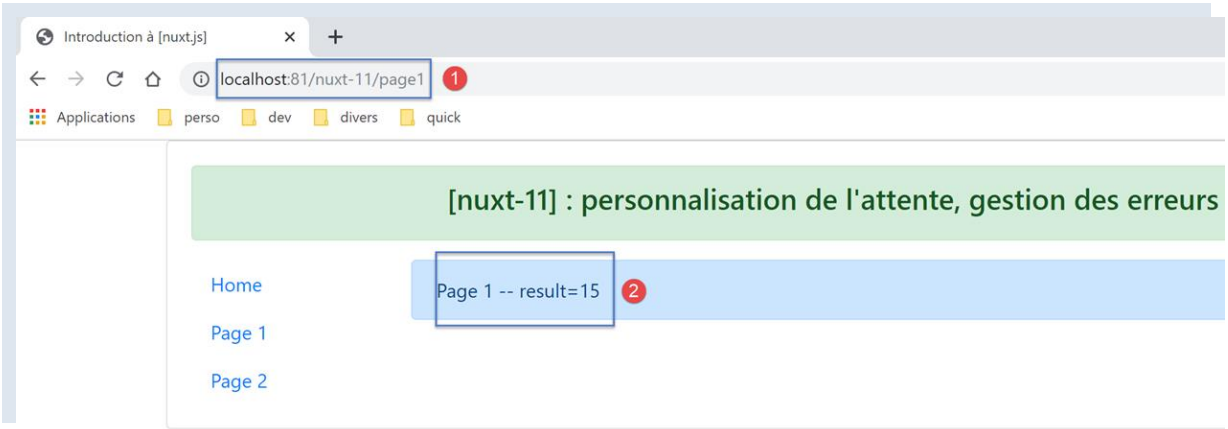


- en **[1]**, la fonction **[asyncData]** démarre ;
- en **[2]**, on met en route l'image d'attente ;
- en **[2-3]**, on voit que la page **[default]** a reçu les événements **[loading, true] [2]** et **[errorLoading, false]** envoyés par la fonction **[asyncData]** de la page **[index]** (lignes 5 et 7);
- en **[4]**, fin de l'attente. La page **[default]** a reçu l'événement **[loading, false]** envoyé par la page **[index]** (ligne 13);
- en **[5]**, la fonction **[asyncData]** a terminé son travail ;

- parce que la fonction **[asyncData]** a créé une erreur avec **[context.error]** (ligne 20), la page **[error]** est affichée **[6]**;

4.14.6.6 La page [page1] exécutée par le client

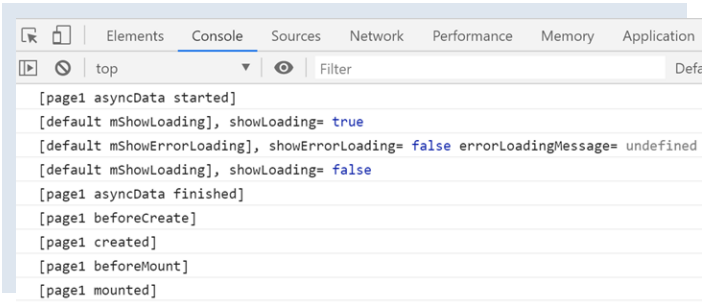
Après l'attente de 5 secondes, le client affiche la page suivante :



Rappelons le code de la fonction **[asyncData]** de **[page1]** :

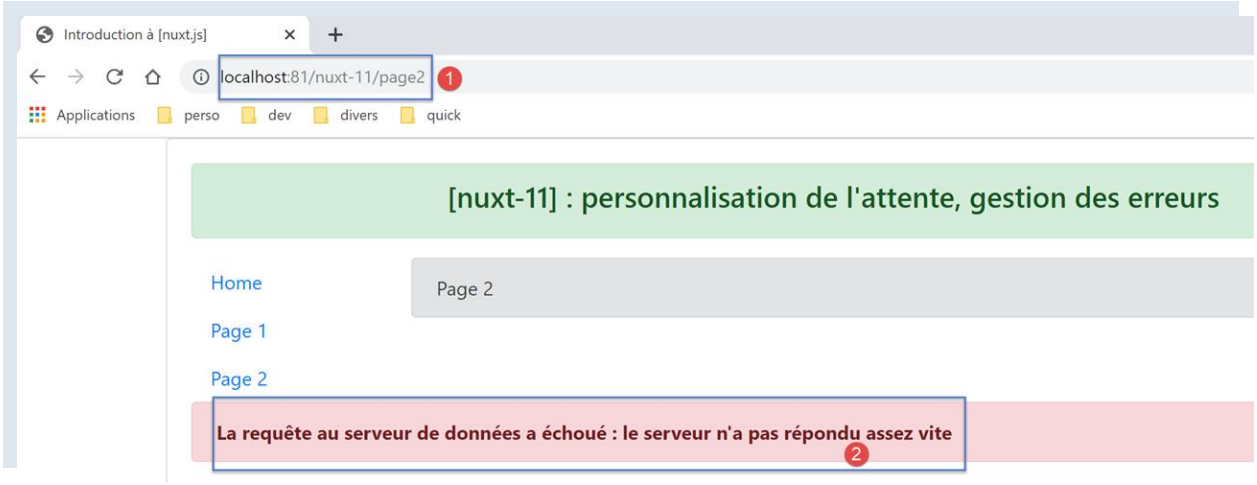
```
1.    // données asynchrones
2.    asyncData(context) {
3.      // log
4.      console.log('[page1 asyncData started]')
5.      // début attente
6.      context.app.$eventBus().$emit('loading', true)
7.      // pas d'erreur
8.      context.app.$eventBus().$emit('errorLoading', false)
9.      // on rend une promesse
10.     return new Promise(function(resolve, reject) {
11.       // on simule une fonction asynchrone
12.       setTimeout(function() {
13.         // fin attente
14.         context.app.$eventBus().$emit('loading', false)
15.         // log
16.         console.log('[page1 asyncData finished]')
17.         // on rend le résultat asynchrone - un nombre aléatoire ici
18.         resolve({ result: Math.floor(Math.random() * Math.floor(100)) })
19.       }, 5000)
20.     })
21.   },
```

Les logs sont les suivants :



4.14.6.7 La page [page2] exécutée par le client

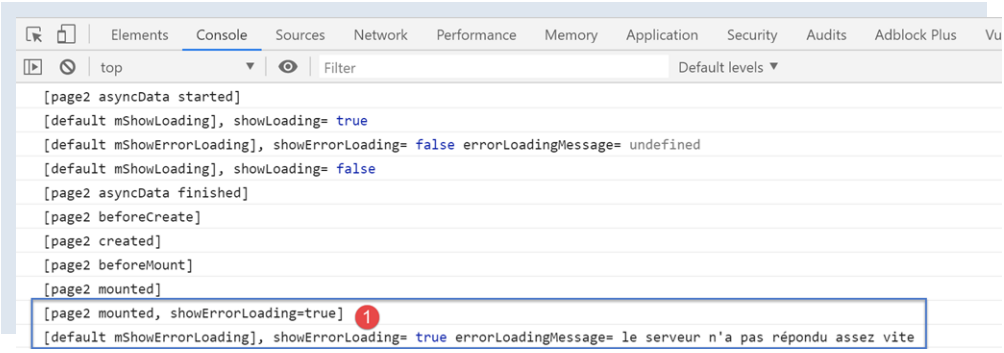
Après l'attente de 5 secondes, le client affiche la page suivante :



Rappelons le code des fonctions **[asyncData]** et **[mounted]** de **[page2]** :

```
1.    // données asynchrones
2.    asyncData(context) {
3.      // log
4.      console.log('[page2 asyncData started]')
5.      // début attente
6.      context.app.$eventBus().$emit('loading', true)
7.      // pas d'erreur
8.      context.app.$eventBus().$emit('errorLoading', false)
9.      // on rend une promesse
10.     return new Promise(function(resolve, reject) {
11.       // on simule une fonction asynchrone
12.       setTimeout(function() {
13.         // fin attente
14.         context.app.$eventBus().$emit('loading', false)
15.         // on génére arbitrairement une erreur
16.         const errorLoadingMessage = "le serveur n'a pas répondu assez vite"
17.         // fin avec succès
18.         resolve({ showErrorLoading: true, errorLoadingMessage })
19.         // log
20.         console.log('[page2 asyncData finished]')
21.       }, 5000)
22.     })
23.   },
24.   // cycle de vie
25.   …
26.   mounted() {
27.     console.log('[page2 mounted]')
28.     // client
29.     if (this.showErrorLoading) {
30.       console.log('[page2 mounted, showErrorLoading=true]')
31.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
32.     }
33.   }
```

Les logs sont les suivants :



- en **[1]**, la page **[default]** a reçu l'événement **[showErrorLoading, true]** envoyé par **[page2]** (ligne 31) qui lui demande d'afficher le message d'erreur ;

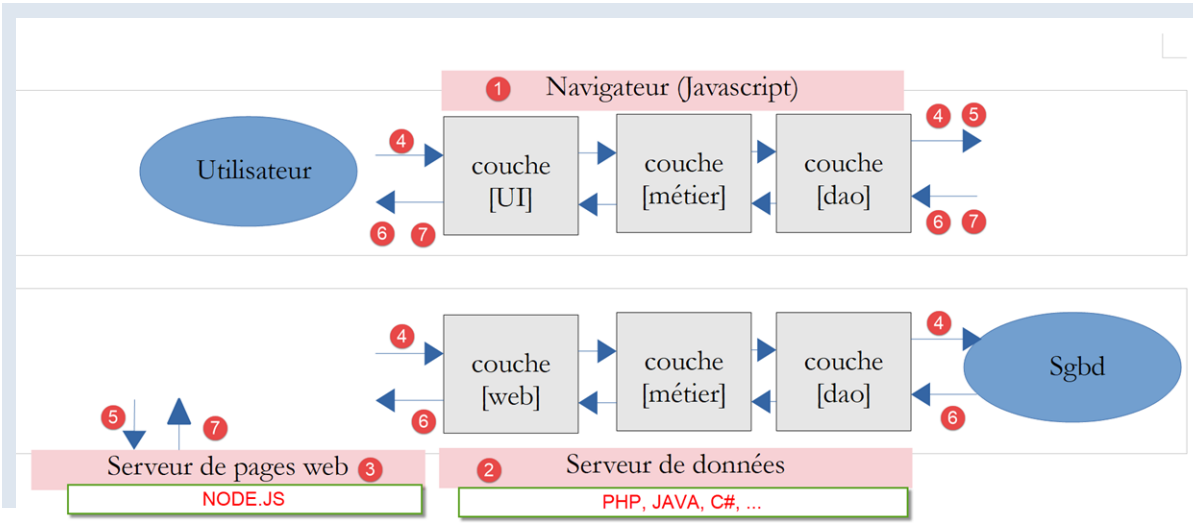
4.15 Exemple [nuxt-12] : requêtes HTTP avec axios

4.15.1 Présentation

Dans ce nouvel exemple, nous allons découvrir comment dans les fonctions **[asyncData]** on peut faire des requêtes HTTP avec la bibliothèque **[axios]**. Par ailleurs, nous allons utiliser des notions déjà acquises :

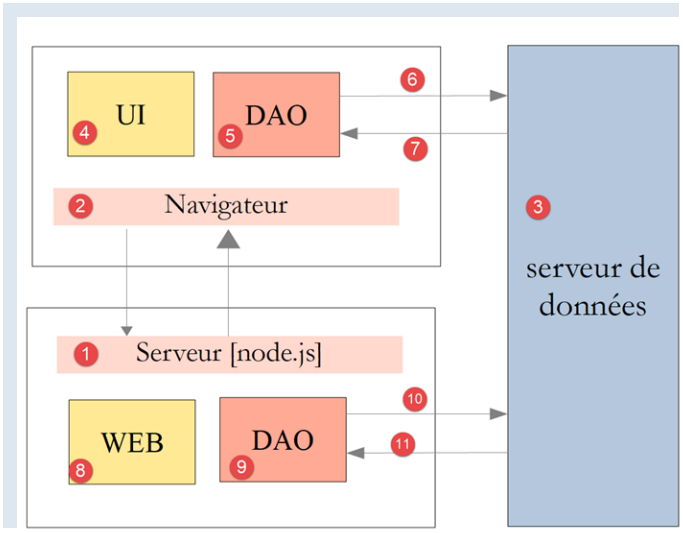
- l'utilisation de plugins de l'exemple **[nuxt-06]** :
- la persistance du store dans un cookie de session de l'exemple **[nuxt-06]** ;
- le contrôle de la navigation avec des middlewares de l'exemple **[nuxt-09]** ;
- la gestion des erreurs de l'exemple **[nuxt-11]** ;

L'architecture de l'exemple sera le suivant :



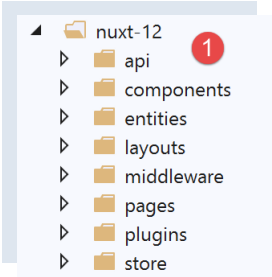
- l'application **[nuxt]** sera stockée sur le serveur **[node.js] [3]**, téléchargée par le navigateur **[1]** qui l'exécutera ensuite ;
- le client **[nuxt] [1]** aussi bien que le serveur **[nuxt] [3]** feront des requêtes HTTP au serveur de données **[2]**. Ce serveur sera le serveur de calcul de l'impôt développé dans la partie PHP 7. Nous utiliserons sa dernière version, la version 14, avec les requêtes CORS autorisées ;

L'architecture de l'exemple peut être simplifiée de la façon suivante :



- en **[1]**, le serveur **[node.js]** délivre les pages **[nuxt]** au navigateur **[2]**. C'est la couche **[web] [8]** du serveur qui délivre ces pages. Pour délivrer la page, le serveur a pu demander des données externes au serveur de données **[3]**. C'est la couche **[DAO] [9]** qui fait les requêtes HTTP nécessaires ;
- à chaque appel de page au serveur **[node.js][1]**, le navigateur **[2]** reçoit la totalité de l'application **[nuxt]** qui va alors s'exécuter en mode SPA. Le bloc **[UI]** (User Interface) **[4]** présente des pages **[vue.js]** à l'utilisateur. Les actions de celui-ci ou le cycle de vie naturel des pages peuvent provoquer des appels de données externes au serveur de données **[3]**. C'est la couche **[DAO] [5]** qui fait alors les requêtes HTTP nécessaires ;

4.15.2 Arborescence du projet



4.15.3 Le fichier de configuration [nuxt.config.js]

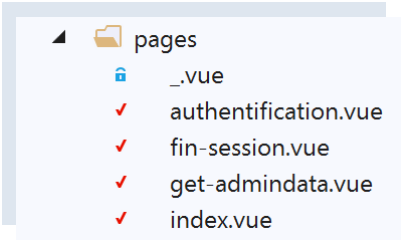
Le projet sera contrôlé par le fichier **[nuxt.config.js]** suivant :

```
1.  export default {
2.    mode: 'universal',
3.    /*
4.     ** Headers of the page
5.     */
6.    head: {
7.      title: 'Introduction à [nuxt.js]',
8.      meta: [
9.        { charset: 'utf-8' },
10.       { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.       {
12.         hid: 'description',
13.         name: 'description',
14.         content: 'ssr routing loading asyncdata middleware plugins store'
15.       }
16.     ],
17.     link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.   },
19.   /*
20.    ** Customize the progress-bar color
21.    */
22.   loading: false,
23.
24.   /*
25.    ** Global CSS
26.    */
27.   css: [],
28.   /*
29.    ** Plugins to load before mounting the App
30.    */
31.   plugins: [
32.     { src: '@/plugins/client/plgSession', mode: 'client' },
33.     { src: '@/plugins/server/plgSession', mode: 'server' },
34.     { src: '@/plugins/client/plgDao', mode: 'client' },
35.     { src: '@/plugins/server/plgDao', mode: 'server' },
36.     { src: '@/plugins/client/plgEventBus', mode: 'client' }
37.   ],
38.   /*
39.    ** Nuxt.js dev-modules
40.    */
41.   buildModules: [
42.     // Doc: https://github.com/nuxt-community/eslint-module
43.     '@nuxtjs/eslint-module'
44.   ],
45.   /*
46.    ** Nuxt.js modules
47.    */
```

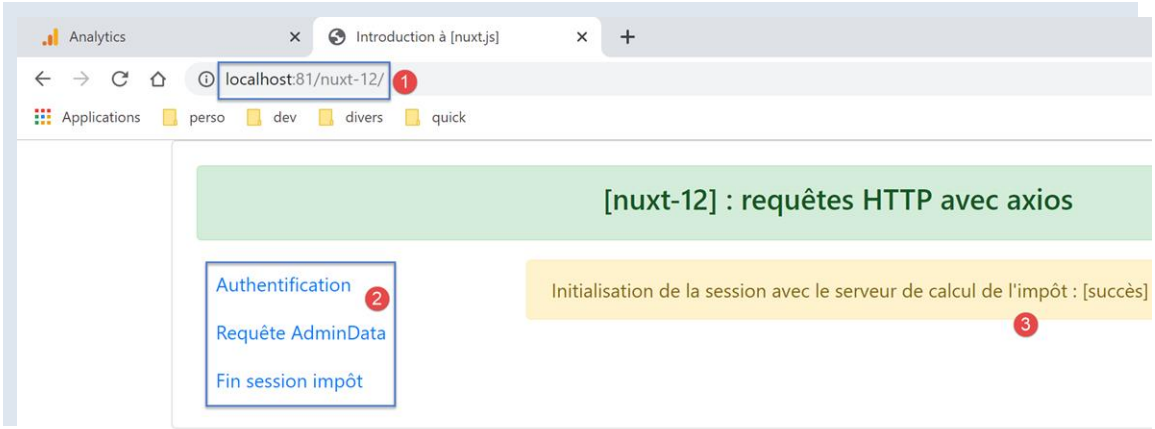
```
48.   modules: [
49.     // Doc: https://bootstrap-vue.js.org
50.     'bootstrap-vue/nuxt',
51.     // Doc: https://axios.nuxtjs.org/usage
52.     '@nuxtjs/axios',
53.     // https://www.npmjs.com/package/cookie-universal-nuxt
54.     'cookie-universal-nuxt'
55.   ],
56.   /*
57.    ** Axios module configuration
58.    ** See https://axios.nuxtjs.org/options
59.    */
60.   axios: {},
61.   /*
62.    ** Build configuration
63.    */
64.   build: {
65.     /*
66.      ** You can extend webpack config here
67.      */
68.     extend(config, ctx) {}
69.   },
70.   // répertoire du code source
71.   srcDir: 'nuxt-12',
72.   // routeur
73.   router: {
74.     // racine des URL de l'application
75.     base: '/nuxt-12/',
76.     // middleware de routage
77.     middleware: ['routing']
78.   },
79.   // serveur
80.   server: {
81.     // port de service, 3000 par défaut
82.     port: 81,
83.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
84.     // 0.0.0.0 = toutes les adresses réseau de la machine
85.     host: 'localhost'
86.   },
87.   // environnement
88.   env: {
89.     // configuration axios
90.     timeout: 2000,
91.     withCredentials: true,
92.     baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
93.     // configuration du cookie de session [nuxt]
94.     maxAge: 60 * 5
95.   }
96. }
```

- ligne 22 : nous gérons nous-mêmes l'alerte d'attente de fin d'une action asynchrone ;
- ligne 31 : nous allons utiliser divers plugins qui seront spécialisés soit pour le client soit pour le serveur mais pas pour les deux à la fois ;
- ligne 52 : le module **[axios]** est intégré à **[nuxt]**. Cela va avoir pour conséquence que l'objet **[axios]** qui fera les requêtes HTTP de l'application **[nuxt]** vers le serveur PHP de calcul de l'impôt sera disponible dans **[context.$axios]** ;
- ligne 54 : le module **[cookie-universal-nuxt]** va nous permettre de sauvegarder la session **[nuxt]** dans un cookie ;
- ligne 60 : la propriété **[axios]** nous permet de configurer le module **[@nuxtjs/axios]** de la ligne 52. Nous n'utiliserons pas cette possibilité à laquelle nous préfèrerons la propriété **[env]** de la ligne 88 ;
- ligne 90 : durée d'attente maximale de la réponse du serveur de calcul de l'impôt ;
- ligne 91 : nécessaire au client **[nuxt]** - autorise l'utilisation de cookies dans les échanges avec le serveur de calcul de l'impôt ;
- ligne 92 : l'URL de base du serveur de calcul de l'impôt ;
- ligne 94 : durée de vie de la session nuxt (5 mn) ;
- ligne 77 : la navigation des client et serveur **[nuxt]** sera contrôlée par un middleware de routage ;

4.15.4 La couche [UI] de l'application



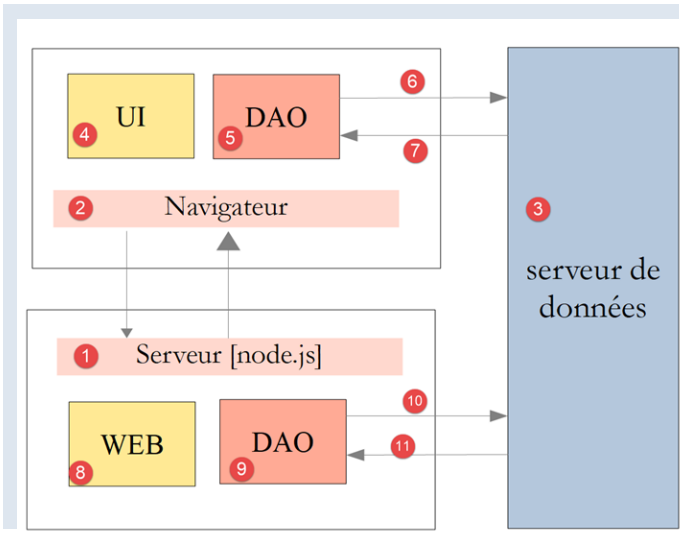
Nous allons donner à l'application **[nuxt]** l'accès à l'API du serveur de calcul de l'impôt via la vue suivante :



- en **[2]**, le menu qui donne accès à l'API du serveur de calcul de l'impôt :
  - **[Authentification]** : correspond à la page **[authentification]**. Cette page fait une requête d'authentification auprès du serveur de calcul de l'impôt avec les identifiants **[admin, admin]** qui sont pour l'instant les seuls autorisés. Le résultat affiché est analogue à **[3]** ;
  - **[Requête AdminData]** : correspond à la page **[get-admindata]**. Cette page demande au serveur de calcul de l'impôt, les données, appelées ici **[adminData]**, qui permettent le calcul de l'impôt. Le résultat affiché est analogue à **[3]** ;
  - **[Fin session impôt]** : correspond à la page **[fin-session]**. Cette page fait une requête de fin de session PHP auprès du serveur de calcul de l'impôt. Le serveur annule alors la session PHP courante et en initialise une nouvelle vierge ;

4.15.5 Les couches [dao] de l'application [nuxt]

Comme indiqué plus haut, l'architecture de l'application **[nuxt]** sera la suivante :



- en **[1]**, le serveur **[node.js]** délivre les pages **[nuxt]** au navigateur **[2]**. C'est la couche **[web] [8]** du serveur qui délivre ces pages. Pour délivrer la page, le serveur a pu demander des données externes au serveur de données **[3]**. C'est la couche **[DAO] [9]** qui fait les requêtes HTTP nécessaires ;
- à chaque appel de page au serveur **[node.js][1]**, le navigateur **[2]** reçoit la totalité de l'application **[nuxt]** qui va alors s'exécuter en mode SPA. Le bloc **[UI]** (User Interface) **[4]** présente des pages **[vue.js]** à l'utilisateur. Les actions de celui-ci ou le cycle de vie des pages peuvent provoquer des appels de données externes au serveur de données **[3]**. C'est la couche **[DAO] [5]** qui fait alors les requêtes HTTP nécessaires ;

Nous utiliserons la version 14 du serveur de calcul de l'impôt. Nous utiliserons une partie seulement de son API (**A**pplication **P**rogramming **I**nterface) jSON :

Requête	Réponse

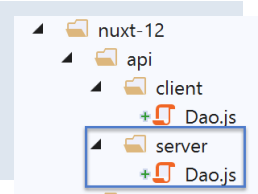
[initialisation d'une session jSON avec le serveur de calcul de l'impôt]
[une session PHP est créée avec le serveur de calcul de l'impôt]
 	
`GET main.php?action=init-session&type=json`
	`{`
	`  "action": "init-session",`
	`  "état": 700,`
	`  "réponse": "session démarrée avec type [json]"`
	`}`

[authentification de l'utilisateur]
[l'authentification est stockée dans la session PHP]
 	
`POST main.php?action=authentifier-utilisateur`
`paramètres postés : user, admin`
	`{`
	`  "action": "authentifier-utilisateur",`
	`  "état": 200,`
	`  "réponse": "Authentification réussie [admin, admin ]"`
	`}`

[demande des données de l'administration fiscale]
[les données reçues sont stockées dans la session PHP]
 	
`GET main.php?action=get-admindata`
	`{`
	`  "action": "get-admindata",`
	`  "état": 1000,`
	`  "réponse": {`
	`  "limites": [`
	`  9964,`
	`  27519,`
	`  73779,`
	`  156244,`
	`  0`
	`  ],`
	`  "coeffR": [`
	`  0,`
	`  0.14,`
	`  0.3,`
	`  0.41,`
	`  0.45`
	`  ],`
	`  "coeffN": [`
	`  0,`
	`  1394.96,`
	`  5798,`
	`  13913.69,`
	`  20163.45`
	`  ],`
	`  "plafondQfDemiPart": 1551,`
	`  "plafondRevenusCelibatairePourReduction": 21037,`
	`  "plafondRevenusCouplePourReduction": 42074,`
	`  "valeurReducDemiPart": 3797,`
	`  "plafondDecoteCelibataire": 1196,`
	`  "plafondDecoteCouple": 1970,`
	`  "plafondImpotCouplePourDecote": 2627,`
	`  "plafondImpotCelibatairePourDecote": 1595,`
	`  "abattementDixPourcentMax": 12502,`
	`  "abattementDixPourcentMin": 437`
	`  }`
	`}`

[fin de la session PHP avec le serveur de calcul de l'impôt]
[supprime la session PHP courante et crée une nouvelle session PHP. Dans celle-ci, la session jSON reste activée, mais l'utilisateur n'est plus authentifié]
 	
`GET main.php?action=fin-session`
	`{`
	`  "action": "fin-session",`
	`  "état": 400,`
	`  "réponse": "session supprimée"`
	`}`

4.15.5.1 Le couche [dao] du serveur [nuxt]



Le serveur **[node.js] [1]** va utiliser la couche **[dao]** décrite dans le document |Introduction à ECMASCRIPT 6 par l'exemple|. Nous rappelons ici son code :

```
1.  'use strict';
2.
3.  // imports
4.  import qs from 'qs'
5.
6.  class Dao2 {
7.
8.    // constructeur
9.    constructor(axios) {
10.     this.axios = axios;
11.     // cookie de session
12.     this.sessionCookieName = "PHPSESSID";
13.     this.sessionCookie = '';
14.   }
15.
16.   // init session
17.   async initSession() {
18.     // options de la requête HHTP [get /main.php?action=init-session&type=json]
19.     const options = {
20.       method: "GET",
21.       // paramètres de l'URL
22.       params: {
23.         action: 'init-session',
24.         type: 'json'
25.       }
26.     };
27.     // exécution de la requête HTTP
28.     return await this.getRemoteData(options);
29.   }
30.
31.   async authentifierUtilisateur(user, password) {
32.     // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
33.     const options = {
34.       method: "POST",
35.       headers: {
36.         'Content-type': 'application/x-www-form-urlencoded',
37.       },
38.       // corps du POST
39.       data: qs.stringify({
40.         user: user,
41.         password: password
42.       }),
43.       // paramètres de l'URL
44.       params: {
45.         action: 'authentifier-utilisateur'
46.       }
47.     };
48.     // exécution de la requête HTTP
49.     return await this.getRemoteData(options);
50.   }
51.
52.   async getAdminData() {
53.     // options de la requête HHTP  [get /main.php?action=get-admindata]
54.     const options = {
55.       method: "GET",
56.       // paramètres de l'URL
57.       params: {
58.         action: 'get-admindata'
59.       }
```

```
60.     };
61.     // exécution de la requête HTTP
62.     const data = await this.getRemoteData(options);
63.     // résultat
64.     return data;
65.   }
66.
67.   async getRemoteData(options) {
68.     // pour le cookie de session
69.     if (!options.headers) {
70.       options.headers = {};
71.     }
72.     options.headers.Cookie = this.sessionCookie;
73.     // exécution de la requête HTTP
74.     let response;
75.     try {
76.       // requête asynchrone
77.       response = await this.axios.request('main.php', options);
78.     } catch (error) {
79.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
80.       if (error.response) {
81.         // la réponse du serveur est dans [error.response]
82.         response = error.response;
83.       } else {
84.         // on relance l'erreur
85.         throw error;
86.       }
87.     }
88.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
89.     // on récupère le cookie de session s'il existe
90.     const setCookie = response.headers['set-cookie'];
91.     if (setCookie) {
92.       // setCookie est un tableau
93.       // on cherche le cookie de session dans ce tableau
94.       let trouvé = false;
95.       let i = 0;
96.       while (!trouvé && i < setCookie.length) {
97.         // on cherche le cookie de session
98.         const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
99.         if (results) {
100.          // on mémorise le cookie de session
101.          // eslint-disable-next-line require-atomic-updates
102.          this.sessionCookie = results[1];
103.          // on a trouvé
104.          trouvé = true;
105.        } else {
106.          // élément suivant
107.          i++;
108.        }
109.      }
110.    }
111.    // la réponse du serveur est dans [response.data]
112.    return response.data;
113.  }
114.}
115.
116.// export de la classe
117.export default Dao2;
```

- toutes les méthodes de la couche **[dao]** rendent l'objet envoyé par le serveur de données **[{action : 'xx', état : nn, réponse : {...}]** avec :
  - **[action]** : le nom de l'action exécutée par le serveur de données ;
  - **[état]** : indicateur numérique :
    - **[initSession]** : **état**=700 pour une réponse sans erreur ;
    - **[authentifierUtilisateur]** : **état**=200 pour une réponse sans erreur ;
    - **[getAdminData]** : **état**=1000 pour une réponse sans erreur ;
    - **[fin-session]** : **état**=400 pour une réponse sans erreur ;
  - **[réponse]** : réponse associée à l'indicateur numérique **[état]**. Peut varier selon cet indicateur numérique ;

Examinons le constructeur de la classe **[Dao]** :

```
1.    // constructeur
2.    constructor(axios) {
3.      this.axios = axios;
4.      // cookie de session
```

```
5.      this.sessionCookieName = "PHPSESSID";
6.      this.sessionCookie = '';
7.    }
```

- ligne 2 : l'objet **[axios]** fourni en argument au constructeur est fourni par le code appelant. C'est lui qui va faire les requêtes HTTP ;
- ligne 5 : le nom du cookie de session envoyé par le serveur de données écrit en PHP ;
- ligne 6 : le cookie de session qui est échangé entre la couche **[dao]** et le serveur de données. Celui-ci est initialisé par la fonction **[getRemoteData]** des lignes 67-113 ;

Pour le cookie de session, il nous faut considérer deux couches **[dao]** séparées :

- celle du navigateur ;
- celle du serveur ;

Nous allons devoir gérer trois cookies de session :

- celui échangé entre le client **[nuxt]** et le serveur PHP 7 ;
- celui échangé entre le serveur **[nuxt]** et le serveur PHP 7 ;
- celui échangé entre le client **[nuxt]** et le serveur **[nuxt]** ;

Nous ferons en sorte que le cookie de la session avec le serveur PHP soit le même pour le client et le serveur **[nuxt]**. Nous appellerons ce cookie, **cookie de la session PHP**. Ce cookie est celui des cas 1 et 2. Nous appellerons **cookie de la session [nuxt]**, le cookie du cas 3. Nous aurons donc deux sessions :

- une session PHP avec le cookie de session PHP ;
- une session **[nuxt]** avec le cookie de session **[nuxt]** ;

Pourquoi utiliser le même cookie pour les sessions PHP du client et du navigateur **[nuxt]** ? Nous voulons que l'application puisse dialoguer avec le serveur PHP 7 indifféremment avec le client ou le serveur **[nuxt]** :

- si une action A du serveur **[nuxt]** met le serveur PHP dans un état E, cet état est reflété dans la session PHP entretenue par le serveur PHP ;
- en utilisant le même cookie de session PHP que le serveur, une action B du client **[nuxt]** qui suivrait l'action A du serveur **[nuxt]** retrouverait le serveur PHP dans l'état E laissé par le serveur **[nuxt]** et pourrait donc s'appuyer sur le travail déjà fait par le serveur **[nuxt]** ;
- si après l'action B du client **[nuxt]**, vient une action C du serveur **[nuxt]**, pour la même raison que précédemment, cette action va pouvoir s'appuyer sur le travail fait par l'action B du client **[nuxt]** ;

Pour que le navigateur du client **[nuxt]** puisse dialoguer avec le serveur PHP du calcul de l'impôt, nous utiliserons la version 14 de ce serveur **qui autorise les appels inter-domaines**, ç-à-d ceux d'un navigateur vers le serveur PHP. Les appels du serveur **[nuxt]** vers le serveur PHP ne sont pas, eux, des appels inter-domaines. Cette notion n'existe que pour les appels faits depuis un navigateur.

Revenons au code du constructeur de la classe **[Dao]** précédente :

```
1.    // constructeur
2.    constructor(axios) {
3.      this.axios = axios;
4.      // cookie de session
5.      this.sessionCookieName = "PHPSESSID";
6.      this.sessionCookie = '';
7.    }
```

- les lignes 5 et 6 correspondent au cookie de la session PHP avec le serveur de calcul de l'impôt ;

La gestion du cookie de la session PHP ci-dessus ne convient pas au serveur **[nuxt]** : sa couche **[dao] est instanciée à chaque nouvelle requête faite au serveur [nuxt]**. On se rappelle en effet que demander une page au serveur **[nuxt]** revient à réinitialiser l'application **[nuxt]**. Ainsi lorsqu'à l'issue de la 1ère requête faite au serveur de données par le serveur **[nuxt]**, le cookie de session PHP de la couche **[dao]** est initialisé, cette valeur est perdue lors de la requête HTTP suivante du même serveur **[nuxt]**, car entre-temps sa couche **[dao]** a été recréée, le constructeur réexécuté et le cookie de session PHP réinitialisé avec la chaîne vide (ligne 6) ;

Une solution est d'utiliser un autre constructeur pour la couche **[dao]** du serveur :

```
1.  // constructeur
2.    constructor(axios, phpSessionCookie) {
3.      // bibliothèque axios
4.      this.axios = axios
```

```
5.      // valeur du cookie de session
6.      this.phpSessionCookie = phpSessionCookie
7.      // nom du cookie de session du serveur PHP
8.      this.phpSessionCookieName = 'PHPSESSID'
9.  }
```

- ligne 2 : cette fois-ci le cookie de la session PHP sera fourni au constructeur de la couche **[dao]** du serveur de données ;

Comment le serveur **[nuxt]** pourra-t-il fournir ce cookie de session PHP au constructeur de sa couche **[dao]** ? Nous stockerons le cookie de session PHP dans le cookie de session **[nuxt]** échangé entre le navigateur et le serveur **[nuxt]**. Le processus est le suivant :

- l'application **[nuxt]** est lancée ;
- lorsque le serveur **[nuxt]** fait sa 1ère requête HTTP avec le serveur PHP, il stocke le cookie de la session PHP qu'il a reçu dans le cookie de session **[nuxt]** qu'il échange avec le client **[nuxt]** ;
- le navigateur qui loge le client **[nuxt]** reçoit ce cookie de session **[nuxt]** et le renvoie donc systématiquement à chaque nouvelle requête au serveur **[nuxt]** ;
- lorsque le serveur **[nuxt]** devra faire une nouvelle requête au serveur PHP, il retrouvera le cookie de session PHP dans le cookie de session **[nuxt]** que le navigateur lui aura envoyé. Il l'enverra alors au serveur PHP ;

Il y a bien deux cookies de session et il ne faut pas les confondre :

- le cookie de session **[nuxt]** échangé entre le serveur **[nuxt]** et le navigateur du client **[nuxt]** ;
- le cookie de session PHP échangé entre le serveur **[nuxt]** et le serveur PHP ou entre le client **[nuxt]** et le serveur PHP ;

Revenons maintenant sur le code de la méthode de la classe **[Dao]**. Elle n'inclut pas de fonction pour clôre la session PHP avec le serveur de calcul de l'impôt. Nous ajoutons celle-ci :

```
1.    // fin de la session de calcul de l'impôt
2.    async finSession() {
3.      // options de la requête HHTP  [get /main.php?action=fin-session]
4.      const options = {
5.        method: 'GET',
6.        // paramètres de l'URL
7.        params: {
8.          action: 'fin-session'
9.        }
10.     }
11.     // exécution de la requête HTTP
12.     const data = await this.getRemoteData(options)
13.     // résultat
14.     return data
15.   }
```

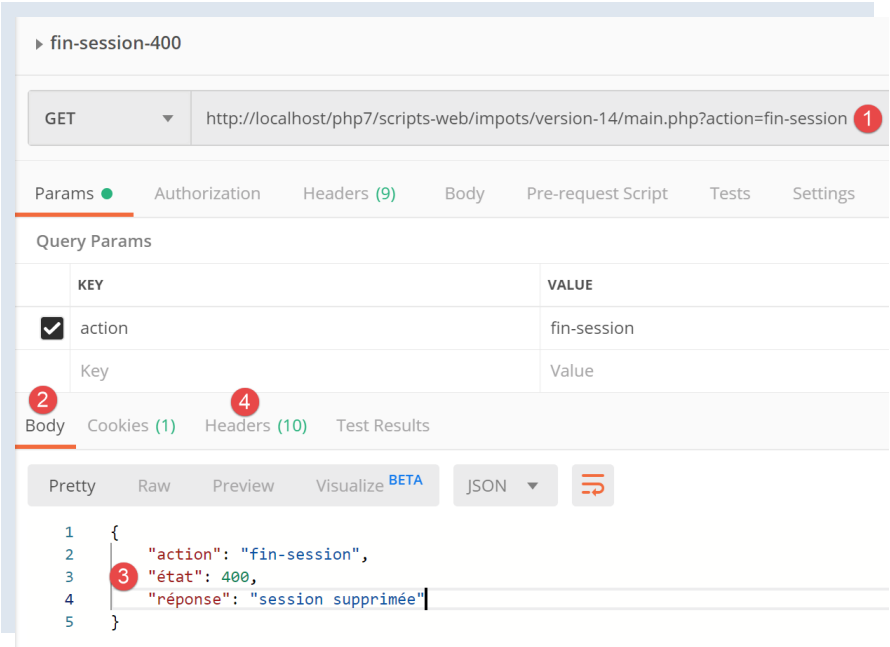
Aux tests, on découvre que la fonction **[getRemoteData]** appelée ligne 12 ne convient pas pour la méthode **[finSession]** :

```
1.    async getRemoteData(options) {
2.      // pour le cookie de session
3.      if (!options.headers) {
4.        options.headers = {};
5.      }
6.      options.headers.Cookie = this.sessionCookie;
7.      // exécution de la requête HTTP
8.      let response;
9.      try {
10.       // requête asynchrone
11.       response = await this.axios.request('main.php', options);
12.     } catch (error) {
13.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
14.       if (error.response) {
15.         // la réponse du serveur est dans [error.response]
16.         response = error.response;
17.       } else {
18.         // on relance l'erreur
19.         throw error;
20.       }
21.     }
22.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
23.     // on récupère le cookie de session s'il existe
24.     const setCookie = response.headers['set-cookie'];
25.     if (setCookie) {
26.       // setCookie est un tableau
27.       // on cherche le cookie de session dans ce tableau
```

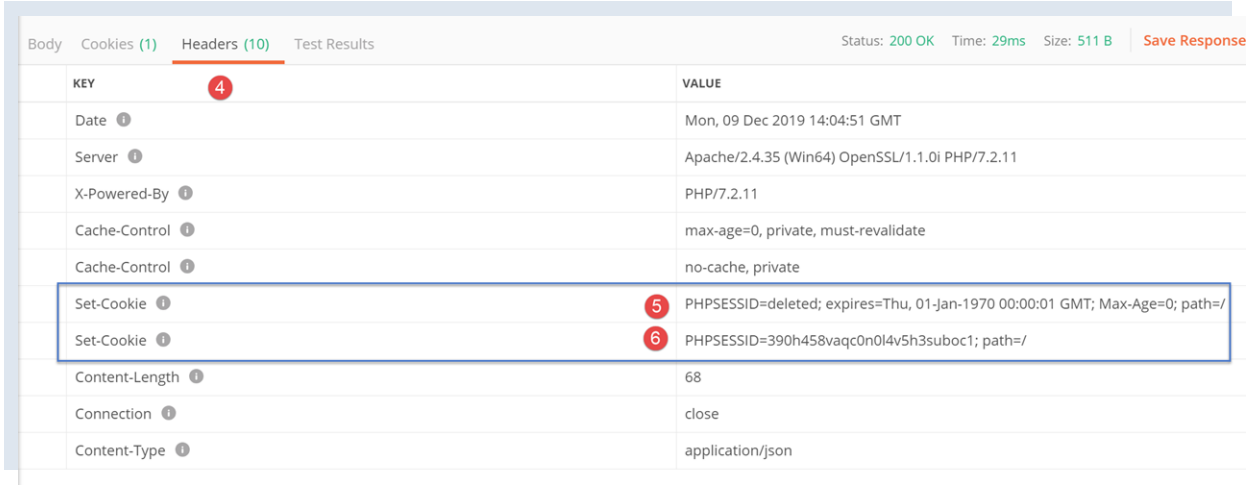
```
28.       let trouvé = false;
29.       let i = 0;
30.       while (!trouvé && i < setCookie.length) {
31.         // on cherche le cookie de session
32.         const results = RegExp('^(' + this.sessionCookieName + '.+?);').exec(setCookie[i]);
33.         if (results) {
34.           // on mémorise le cookie de session
35.           // eslint-disable-next-line require-atomic-updates
36.           this.sessionCookie = results[1];
37.           // on a trouvé
38.           trouvé = true;
39.         } else {
40.           // élément suivant
41.           i++;
42.         }
43.       }
44.     }
45.     // la réponse du serveur est dans [response.data]
46.     return response.data;
47.   }
```

- lignes 30-43 : on recherche le cookie **[PHPSESSID=xxx]**. Si on le trouve, il est mémorisé dans la classe (ligne 36) ;

Ce code ne convient pas à la nouvelle méthode **[finSession]** car sur l'action **[fin-session]**, le serveur PHP envoie **deux** cookies avec le nom **[PHPSESSID]**. Voici un exemple obtenu avec un client **[Postman]** :



- en **[1]**, la demande du client **[Postman]** ;
- en **[3]**, la réponse du serveur PHP ;
- en **[4]**, les entêtes HTTP de la réponse du serveur PHP ;



- en **[5]**, le serveur PHP indique d'abord qu'il a supprimé la session PHP courante ;
- en **[6]**, le serveur PHP envoie le cookie de la nouvelle session PHP ;

Avec le code actuel, la fonction **[getRemoteData]** récupère le cookie **[5]** alors que c'est le cookie **[6]** qu'il faut mémoriser.

Il faut donc faire évoluer le code de la fonction **[getRemoteData]** :

```
1.    async getRemoteData(options) {
2.      // y-a-t-il un cookie de session PHP ?
3.      if (this.phpSessionCookie) {
4.        // y-a-t-il des entêtes ?
5.        if (!options.headers) {
6.          // on crée un objet vide
7.          options.headers = {}
8.        }
9.        // entête du cookie de session PHP
10.       options.headers.Cookie = this.phpSessionCookie
11.     }
12.     // exécution de la requête HTTP
13.     let response
14.     try {
15.       // requête asynchrone
16.       response = await this.axios.request('main.php', options)
17.     } catch (error) {
18.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
19.       if (error.response) {
20.         // la réponse du serveur est dans [error.response]
21.         response = error.response
22.       } else {
23.         // on relance l'erreur
24.         throw error
25.       }
26.     }
27.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
28.     // on cherche le cookie de session PHP dans les cookies reçus
29.     // tous les cookies reçus
30.     const cookies = response.headers['set-cookie']
31.     if (cookies) {
32.       // cookies est un tableau
33.       // on cherche le cookie de session PHP dans ce tableau
34.       let trouvé = false
35.       let i = 0
36.       while (!trouvé && i < cookies.length) {
37.         // on cherche le cookie de session PHP
38.         const results = RegExp('^(' + this.phpSessionCookieName + '.+?)$').exec(cookies[i])
39.         if (results) {
40.           // on mémorise le cookie de session PHP
41.           const phpSessionCookie = results[1]
42.           // y-a-t-il dedans le mot [deleted] ?
43.           const results2 = RegExp(this.phpSessionCookieName + '=deleted').exec(phpSessionCookie)
44.           if (!results2) {
45.             // on a le bon cookie de session PHP
46.             this.phpSessionCookie = phpSessionCookie
47.             // on a trouvé
48.             trouvé = true
```

```
49.           } else {
50.             // élément suivant
51.             i++
52.           }
53.         } else {
54.           // élément suivant
55.           i++
56.         }
57.       }
58.     }
59.     // la réponse du serveur est dans [response.data]
60.     return response.data
61.   }
```

- ligne 41 : on a trouvé un cookie avec le nom **[PHPSESSID]**. on le mémorise localement ;
- ligne 43 : on regarde si dans le cookie sauvegardé, il y a la chaîne **[PHPSESSID=deleted]** ;
- ligne 46 : si la réponse est non, alors c'est qu'on a trouvé le bon cookie **[PHPSESSID]**. On le mémorise dans la classe ;

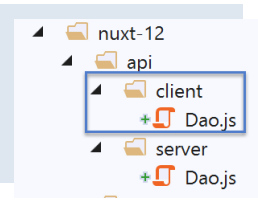
Après la fonction **[getRemoteData]**, le cookie de session PHP est mémorisé dans la classe, dans **[this.phpSessionCookie]**. On a dit que la classe était instanciée à chaque nouvelle requête HTTP du serveur **[nuxt]**. Le cookie de session PHP doit donc être exfiltré de la classe. Pour cela, on ajoute une nouvelle méthode à celle-ci :

```
1.    // accès au cookie de la session PHP
2.    getPhpSessionCookie() {
3.      return this.phpSessionCookie
4.    }
```

- le serveur **[nuxt]** demande une action à sa couche **[dao]** en fournissant le cookie de la session PHP à son constructeur, s'il en a un ;
- une fois l'action faite, le serveur **[nuxt]** récupère le cookie de session PHP mémorisé par la couche **[dao]** à l'aide de la méthode **[getPhpSessionCookie]** précédente. Ce cookie peut être le même que le précédent ou un autre. Ce dernier cas arrive à deux occasions :
  - lors de l'exécution de la méthode **[initSession]** (il n'y avait pas de cookie de session PHP avant) ;
  - lors de l'exécution de la méthode **[finSession]** (le serveur PHP change le cookie de session PHP) ;

Notons une particularité sur le cookie de session PHP. Le serveur **[nuxt]** ne reçoit pas toujours ce cookie de la part du serveur PHP. En effet, celui-ci ne l'envoie qu'une fois. Ensuite il ne l'envoie plus. Lorsqu'on regarde le code de **[getRemoteData]** et celui de **[getPhpSessionCookie]** on verra alors que lorsque le serveur PHP n'envoie pas de cookie de session, la fonction **[getPhpSessionCookie]** renvoie alors le cookie de session PHP fourni au constucteur. C'est ainsi que le serveur envoie toujours au serveur PHP le dernier cookie de session PHP que celui-ci lui a envoyé.

4.15.5.2 La couche [dao] du client [nuxt]



Pour le client **[nuxt]** qui s'exécute dans un navigateur, on reprend le code de la classe **[Dao]** du document | Introduction au framework **VUE.JS** par l'exemple | :

```
1.  "use strict";
2.
3.  // imports
4.  import qs from "qs";
5.
6.  class Dao {
7.    // constructeur
8.    constructor(axios) {
9.      this.axios = axios;
10.   }
11.
12.   // init session
13.   async initSession() {
14.     // options de la requête HHTP [get /main.php?action=init-session&type=json]
15.     const options = {
16.       method: "GET",
```

```
17.       // paramètres de l'URL
18.       params: {
19.         action: "init-session",
20.         type: "json"
21.       }
22.     };
23.     // exécution de la requête HTTP
24.     return await this.getRemoteData(options);
25.   }
26.
27.   async authentifierUtilisateur(user, password) {
28.     // options de la requête HHTP [post /main.php?action=authentifier-utilisateur]
29.     const options = {
30.       method: "POST",
31.       headers: {
32.         "Content-type": "application/x-www-form-urlencoded"
33.       },
34.       // corps du POST
35.       data: qs.stringify({
36.         user: user,
37.         password: password
38.       }),
39.       // paramètres de l'URL
40.       params: {
41.         action: "authentifier-utilisateur"
42.       }
43.     };
44.     // exécution de la requête HTTP
45.     return await this.getRemoteData(options);
46.   }
47.
48.   async getAdminData() {
49.     // options de la requête HHTP  [get /main.php?action=get-admindata]
50.     const options = {
51.       method: "GET",
52.       // paramètres de l'URL
53.       params: {
54.         action: "get-admindata"
55.       }
56.     };
57.     // exécution de la requête HTTP
58.     const data = await this.getRemoteData(options);
59.     // résultat
60.     return data;
61.   }
62.
63.   async getRemoteData(options) {
64.     // exécution de la requête HTTP
65.     let response;
66.     try {
67.       // requête asynchrone
68.       response = await this.axios.request("main.php", options);
69.     } catch (error) {
70.       // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
71.       if (error.response) {
72.         // la réponse du serveur est dans [error.response]
73.         response = error.response;
74.       } else {
75.         // on relance l'erreur
76.         throw error;
77.       }
78.     }
79.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
80.     // la réponse du serveur est dans [response.data]
81.     return response.data;
82.   }
83. }
84.
85. // export de la classe
86. export default Dao;
```

Ce code se distingue de la couche **[dao]** du serveur **[nuxt]** par le fait qu'il ne gère pas le cookie de la session PHP avec le serveur de calcul de l'impôt : c'est le navigateur qui le fait.

Nous allons, comme nous l'avons fait pour la couche **[dao]** du serveur **[nuxt]**, ajouter une méthode **[finSession]** :

```
1.    // fin de la session de calcul de l'impôt
2.    async finSession() {
3.      // options de la requête HHTP  [get /main.php?action=fin-session]
4.      const options = {
5.        method: 'GET',
6.        // paramètres de l'URL
7.        params: {
8.          action: 'fin-session'
9.        }
10.     }
11.     // exécution de la requête HTTP
12.     const data = await this.getRemoteData(options)
13.     // résultat
14.     return data
15.   }
```

Lorsque le client **[nuxt]** exécute cette méthode, il reçoit, comme le serveur **[nuxt]**, deux cookies de session PHP. C'est en fait le navigateur qui les reçoit et il gère correctement la situation : il ne garde que le cookie de la nouvelle session PHP qu'a initiée le serveur de calcul de l'impôt. Donc à la prochaine action du client **[nuxt]** vers le serveur PHP, le cookie de session PHP sera correct car c'est le navigateur qui envoie celui-ci. Il y a cependant un problème : le serveur **[nuxt]** n'a pas connaissance du fait que le cookie de session PHP a changé. Dans ses échanges avec le serveur PHP, il va alors envoyer un cookie de session PHP qui n'existe plus et on va avoir des problèmes. Il faudrait que le client **[nuxt]** avertisse le serveur **[nuxt]** que le cookie de session PHP a changé et lui transmette celui-ci. On sait comment il peut faire cela : via le cookie de session **[nuxt]**, le cookie échangé entre le client et le serveur **[nuxt]**. Le client **[nuxt]** a au moins deux façons de récupérer le nouveau cookie de session PHP :

- en le demandant au navigateur ;
- en utilisant la méthode **[getRemoteData]** du serveur qui sait comment récupérer le nouveau cookie de session PHP ;

Nous allons utiliser la 2ième solution car elle est déjà toute prête. La méthode **[getRemoteData]** du client **[nuxt]** devient alors la suivante :

```
1.    async getRemoteData(options) {
2.      // exécution de la requête HTTP
3.      let response
4.      try {
5.        // requête asynchrone
6.        response = await this.axios.request('main.php', options)
7.      } catch (error) {
8.        // le paramètre [error] est une instance d'exception - elle peut avoir diverses formes
9.        if (error.response) {
10.         // la réponse du serveur est dans [error.response]
11.         response = error.response
12.       } else {
13.         // on relance l'erreur
14.         throw error
15.       }
16.     }
17.     // response est l'ensemble de la réponse HTTP du serveur (entêtes HTTP + réponse elle-même)
18.     // on cherche le cookie de session PHP dans les cookies reçus
19.     // tous les cookies reçus
20.     const cookies = response.headers['set-cookie']
21.     if (cookies) {
22.       // cookies est un tableau
23.       // on cherche le cookie de session PHP dans ce tableau
24.       let trouvé = false
25.       let i = 0
26.       while (!trouvé && i < cookies.length) {
27.         // on cherche le cookie de session PHP
28.         const results = RegExp('^(' + this.phpSessionCookieName + '.+?)$').exec(cookies[i])
29.         if (results) {
30.           // on mémorise le cookie de session PHP
31.           const phpSessionCookie = results[1]
32.           // y-a-t-il dedans le mot [deleted] ?
33.           const results2 = RegExp(this.phpSessionCookieName + '=deleted').exec(phpSessionCookie)
34.           if (!results2) {
35.             // on a le bon cookie de session PHP
36.             this.phpSessionCookie = phpSessionCookie
37.             // on a trouvé
38.             trouvé = true
39.           } else {
40.             // élément suivant
41.             i++
42.           }
43.         } else {
```

```
            // élément suivant
            i++
          }
        }
      }
      // la réponse du serveur est dans [response.data]
      return response.data
    }
```

On a gardé dans **[getRemoteData]** uniquement le code qui exploite la réponse du serveur PHP à la recherche du cookie de session PHP. On n'a pas gardé le code qui incluait le cookie de session PHP dans la requête au serveur PHP car c'est le navigateur qui abrite le client **[nuxt]** qui s'en charge.

Une fois le cookie de session PHP obtenu par le client **[nuxt]**, celui-ci doit être mis dans la session **[nuxt]** pour que le serveur **[nuxt]** puisse en bébéficier. Ce n'est pas la couche **[dao]** qui s'occupe de cela mais elle donne accès par une méthode au cookie de session PHP qu'elle a mémorisé :

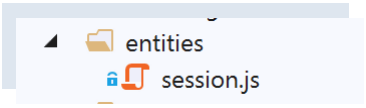
```
    // accès au cookie de la session PHP
    getPhpSessionCookie() {
      return this.phpSessionCookie
    }
```

La fonction **[getPhpSessionCookie]** ne rend pas toujours un cookie de session valide :

- il faut se souvenir ici que la couche **[dao]** du client **[nuxt]** est persistante. Elle est instanciée une fois et reste ensuite en mémoire ;
- tant que le serveur PHP n'envoie pas un cookie de session PHP au client **[nuxt]**, la fonction **[getPhpSessionCookie]** du client **[nuxt]** renvoie une valeur **[undefined]** ;
- lorsque le serveur PHP envoie un cookie de session PHP au client **[nuxt]**, celui-ci est mémorisé dans **[this.phpSessionCookie]** et le restera tant qu'il ne sera pas changé par un nouveau cookie de session PHP envoyé par le serveur PHP. La fonction **[getPhpSessionCookie]** du client **[nuxt]** renvoie alors le dernier cookie de session PHP reçu ;

La couche **[dao]** du client **[nuxt]** ne diffère de celle du serveur **[nuxt]** que par un point : elle n'envoie pas le cookie de session PHP elle-même car c'est le navigateur qui le fait. Néanmoins on a préféré garder deux couches **[dao]** distinctes car les raisonnements qui mènent à leurs écritures respectives sont différents.

4.15.6 La session [nuxt]

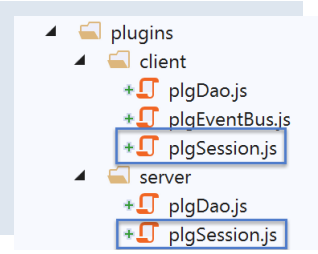


La session **[nuxt]** (entre client et serveur nuxt) sera encapsulée dans l'objet **[session]** suivant :

```
/* eslint-disable no-console */
// définition de la session
const session = {
  // contenu de la session
  value: {
    // store non initialisé
    initStoreDone: false,
    // valeur du store Vuex
    store: ''
  },
  // sauvegarde de la session dans un cookie
  save(context) {
    // sauvegarde du store en session
    this.value.store = context.store.state
    console.log('nuxt-session save=', this.value)
    // sauvegarde de la valeur de la session
    context.app.$cookies.set('nuxt-session', this.value, { path: context.base, maxAge: context.env.maxAge })
  },
  // reset de la session
  reset(context) {
    console.log('nuxt-session reset')
    // reset du store
    context.store.commit('reset')
    // sauvegarde du nouveau store en session et sauvegarde de la session
    this.save(context)
  }
}
// export de la session
export default session
```

- lignes 5-10 : la session n'a qu'une propriété **[value]** avec deux sous-propriétés :
  - **[initStoreDone]** qui indique si le store a été initialisé ou pas ;
  - **[store]** : la valeur **[store.state]** du store Vuex de l'application ;
- lignes 12-18 : la méthode **[save]** sert à sauvegarder la session **[nuxt]** dans un cookie. On utilise ici la bibliothèque **[cookie-universal-nuxt]** pour gérer le cookie. On notera le nom du cookie de la session **[nuxt]** : **[nuxt-session]** (ligne 17) ;
- lignes 20-26 : la méthode **[reset]** réinitialise la session **[nuxt]** ;
  - ligne 23 : le store Vuex est réinitialisé puis sauvegardé en session, ligne 25 ;

4.15.7 Les plugins de gestion de la session [nuxt]



4.15.7.1 Le plugin de gestion de la session [nuxt] du serveur [nuxt]

Au démarrage de l'application, c'est le serveur **[nuxt]** qui opère le premier. C'est donc lui qui va initialiser la session **[nuxt]**. Le script **[server/plgSession]** est le suivant :

```
1.  /* eslint-disable no-console */
2.
3.  // import de la session
4.  import session from '@/entities/session'
5.
6.  export default (context, inject) => {
7.    // gestion de la session serveur
8.    console.log('[plugin server plgSession]')
9.
10.   // y-a-t-il une session existante ?
11.   const value = context.app.$cookies.get('nuxt-session')
12.   if (!value) {
13.     // nouvelle session
14.     console.log("[plugin server plgSession], démarrage d'une nouvelle session")
15.   } else {
16.     // session existante
17.     console.log("[plugin server plgSession], reprise d'une session existante")
18.     session.value = value
19.   }
20.
21.   // on injecte une fonction dans [context, Vue] qui rendra la session courante
22.   inject('session', () => session)
23. }
```

- ligne 4 : on importe le code de la session **[nuxt]** ;
- ligne 11 : on récupère la valeur du cookie de la session **[nuxt]** ;
- lignes 12-15 : si le cookie de la session **[nuxt]** n'existait pas, alors la session **[nuxt]** importée ligne 4 est suffisante. Il n'y a rien de plus à faire ;
- lignes 15-19 : si le cookie de la session **[nuxt]** existait, alors ligne 18 on stocke sa valeur dans la session importée ligne 4 ;
- ligne 22 : la session a été soit initialisée soit restaurée. On la rend disponible via la fonction **[$session]** ;

4.15.7.2 Le plugin de gestion de la session [nuxt] du client [nuxt]

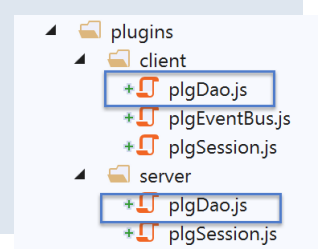
Le script **[client/plgSession]** est le suivant :

```
1.  /* eslint-disable no-console */
2.
3.  // import de la session
4.  import session from '@/entities/session'
5.
6.  export default (context, inject) => {
7.    // gestion de la session client
8.    console.log('[plugin client plgSession], reprise de la session [nuxt] du serveur')
```

```
9.    // on récupère la session existante du serveur nuxt
10.   session.value = context.app.$cookies.get('nuxt-session')
11.
12.   // on injecte une fonction dans [context, Vue] qui rendra la session courante
13.   inject('session', () => session)
14. }
```

- ligne 4 : la session **[nuxt]** est importée ;
- ligne 10 : on récupère la session **[nuxt]** courante dans le cookie **[nuxt-session]** ;
- ligne 13 : on rend la session **[nuxt]** importée ligne 4 au travers de la fonction injectée **[$session]** ;

4.15.8 Les plugins des couches [dao]



4.15.8.1 Le plugin de la couche [dao] du client [nuxt]

Le script **[client/plgDao]** est le suivant :

```
1.  /* eslint-disable no-console */
2.  // on crée un point d'accès à la couche [Dao]
3.  import Dao from '@/api/client/Dao'
4.  export default (context, inject) => {
5.    // configuration axios
6.    context.$axios.defaults.timeout = context.env.timeout
7.    context.$axios.defaults.baseURL = context.env.baseURL
8.    context.$axios.defaults.withCredentials = context.env.withCredentials
9.    // instanciation de la couche [dao]
10.   const dao = new Dao(context.$axios)
11.   // injection d'une fonction [$dao] dans le contexte
12.   inject('dao', () => dao)
13.   // log
14.   console.log('[fonction client $dao créée]')
15. }
```

- ligne 3 : la couche **[dao]** du client **[nuxt]** est importée ;
- lignes 6-8 : on configure l'objet **[context.$axios]** qui va faire les requêtes HTTP de la couche **[dao]** du client **[nuxt]** avec les informations du fichier **[nuxt.config]** :

```
1.    // environnement
2.    env: {
3.      // configuration axios
4.      timeout: 2000,
5.      withCredentials: true,
6.      baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
7.      // configuration du cookie de session [nuxt]
8.      maxAge: 60 * 5
9.    }
```

- ligne 10 : la couche **[dao]** du client **[nuxt]** est instanciée ;
- ligne 12 : la fonction **[$dao]** est injectée dans le contexte et les pages du client. Cette fonction donne accès à la couche **[dao]** de la ligne 10 ;

On retiendra donc que pour avoir accès à la couche **[dao]** du client **[nuxt]** lorsque celui-ci est exécuté, on écrira :

- **[context.app.$dao()]** là où le contexte est connu ;
- **[this.$dao()]** dans une page **[Vue.js]** ;

4.15.8.2 Le plugin de la couche [dao] du *serveur* [nuxt]

Le script **[server/plgDao]** est le suivant :

```
1.  /* eslint-disable no-console */
2.  // on crée un point d'accès à la couche [Dao]
3.  import Dao from '@/api/server/Dao'
4.  export default (context, inject) => {
5.    // configuration axios
6.    context.$axios.defaults.timeout = context.env.timeout
7.    context.$axios.defaults.baseURL = context.env.baseURL
8.    // on récupère le cookie de session
9.    const store = context.app.$session().value.store
10.   const phpSessionCookie = store ? store.phpSessionCookie : ''
11.   console.log('session=', context.app.$session().value, 'phpSessionCookie=', phpSessionCookie)
12.   // instanciation de la couche [dao]
13.   const dao = new Dao(context.$axios, phpSessionCookie)
14.   // injection d'une fonction [$dao] dans le contexte
15.   inject('dao', () => dao)
16.   // log
17.   console.log('[fonction server $dao créée]')
18. }
```

- ligne 3 : la couche **[dao]** du serveur **[nuxt]** est importée ;
- lignes 6-7 : on configure l'objet **[context.$axios]** qui va faire les requêtes HTTP de la couche **[dao]** du serveur **[nuxt]** avec les informations du fichier **[nuxt.config]** :

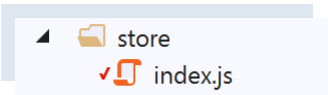
```
1.  // environnement
2.    env: {
3.      // configuration axios
4.      timeout: 2000,
5.      withCredentials: true,
6.      baseURL: 'http://localhost/php7/scripts-web/impots/version-14',
7.      // configuration du cookie de session [nuxt]
8.      maxAge: 60 * 5
9.    }
```

- ligne 9 : on récupère le store de l'application **[nuxt]** ;
- ligne 10 : si le store existe, on récupère le cookie de la session PHP car on en a besoin pour instancier la couche **[dao]** du serveur **[nuxt]** ;
- ligne 13 : on instancie la couche **[dao]** du serveur **[nuxt]** ;
- ligne 15 : la fonction **[$dao]** est injectée dans le contexte et les pages du serveur **[nuxt]**. Cette fonction donne accès à la couche **[dao]** de la ligne 13 ;

On retiendra donc que pour avoir accès à la couche **[dao]** du serveur **[nuxt]** lorsque celui-ci est exécuté, on écrira :

- **[context.app.$dao()]** là où le contexte est connu ;
- **[this.$dao()]** dans une page **[Vue.js]** ;

4.15.9 Le store Vuex



Le store **[Vuex]** va mémoriser toutes les données qui doivent être partagées par les différentes composantes de l'application **[pages, client, serveur]** sans que pour autant ces données soient réactives.

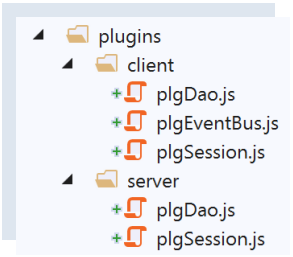
```
1.  /* eslint-disable no-console */
2.
3.  // état du store
4.  export const state = () => ({
5.    // session jSON démarrée
6.    jsonSessionStarted: false,
7.    // utilisateur authentifié
8.    userAuthenticated: false,
9.    // cookie de session PHP
10.   phpSessionCookie: '',
11.   // adminData
12.   adminData: ''
13. })
14.
15. // mutations du store
```

```
16. export const mutations = {
17.   // remplacement du state
18.   replace(state, newState) {
19.     for (const attr in newState) {
20.       state[attr] = newState[attr]
21.     }
22.   },
23.   // reset du store
24.   reset() {
25.     this.commit('replace', { jsonSessionStarted: false, userAuthenticated: false, phpSessionCookie: '', adm
    inData: '' })
26.   }
27. }
28.
29. // actions du store
30. export const actions = {
31.   nuxtServerInit(store, context) {
32.     // qui exécute ce code ?
33.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server, 'env=', context.env)
34.     // init session
35.     initStore(store, context)
36.   }
37. }
38.
39. function initStore(store, context) {
40.   // store est le store à initialiser
41.   // on récupère la session
42.   const session = context.app.$session()
43.   // la session a-t-elle été déjà initialisée ?
44.   if (!session.value.initStoreDone) {
45.     // on démarre un nouveau store
46.     console.log("nuxtServerInit, initialisation d'un nouveau store")
47.     // on met le store dans la session
48.     session.value.store = store.state
49.     // le store est désormais initialisé
50.     session.value.initStoreDone = true
51.   } else {
52.     console.log("nuxtServerInit, reprise d'un store existant")
53.     // on met à jour le store avec le store de la session
54.     store.commit('replace', session.value.store)
55.   }
56.   // on sauvegarde la session
57.   session.save(context)
58.   // log
59.   console.log('initStore terminé, store=', store.state)
60. }
```

Les données mémorisées dans le store sont les suivantes :

- ligne 6 : **[jsonSessionStarted]** sera positionnée à vrai dès que l'initialisation d'une session jSON avec le serveur PHP aura été réussie, qu'elle ait été faite par le client ou le serveur **[nuxt]**. A l'issue de cette initialisation, le cookie de session avec le serveur PHP aura été récupéré et placé dans la propriété **[phpSessionCookie]**, ligne 10 ;
- ligne 8 : **[userAuthenticated]** sera positionnée à vrai dès que l'authentification auprès du serveur PHP aura été réussie, qu'elle ait été faite par le client ou le serveur **[nuxt]** ;
- ligne 12 : **[adminData]** sera la valeur **[adminData]** obtenue auprès du serveur PHP une fois l'authentification réussie ;
- lignes 18-22 : la mutation **[replace]** permet d'initialiser les propriétés précédentes avec celles d'un objet passé en paramètre ;
- lignes 24-26 : la mutation **[reset]** redonne leurs valeurs initiales aux propriétés du store ;
- lignes 31-37 : la fonction **[nuxtServerInit]** délègue son travail à la fonction **[initStore]** ;
- lignes 39-60 : la fonction **[initStore]** a deux rôles :
  - si le store n'a pas été initialisé, il est initialisé et mis en session ;
  - si le store a déjà été initialisé, sa valeur est récupérée dans la session **[nuxt]** ;
- ligne 42 : on récupère la session nuxt ;
- ligne 44 : on regarde si le store a été initialisé :
  - si ce n'est pas le cas, on met le store initial dans la session (ligne 48) ;
  - puis ligne 50, on indique que le store a été initialisé ;
- lignes 51-55 : si le store était initialisé, on utilise alors celui-ci, ligne 54, pour initialiser le store à la valeur contenue dans la session ;
- ligne 57 : dans tous les cas, la session est sauvegardée dans le cookie **[nuxt-session]**, avec le store qu'elle contient ;

4.15.10 Le plugin [plgEventBus]



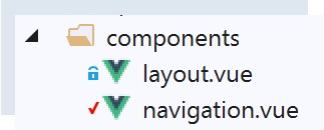
Ce plugin vise à rendre un bus d'événements accessible au client **[nuxt]** via une fonction **[$eventBus]** injectée dans le contexte du client **[nuxt]**. Il est inutile de l'injecter dans le contexte du serveur **[nuxt]** car celui-ci ne sait pas gérer les événements. Néanmoins nous avons déjà vu que l'injecter côté serveur puis l'utiliser ne provoque pas d'erreur.

```
1.  /* eslint-disable no-console */
2.  // on crée un bus d'événements entre les vues
3.  import Vue from 'vue'
4.  export default (context, inject) => {
5.    // le bus d'événement
6.    const eventBus = new Vue()
7.    // injection d'une fonction [eventBus] dans le contexte
8.    inject('eventBus', () => eventBus)
9.    // log
10.   console.log('[fonction $eventBus créée]')
11. }
```

Nous avons déjà rencontré ce plugin au paragraphe lien. La fonction **[$eventBus]** sera disponible au client via les notations :

- **[context.app.$eventBus()]** là où le contexte est disponible ;
- **[this.$eventBus()]** dans les pages **[Vue.js]** du client ;

4.15.11 Les composants de l'application [nuxt]



Le composant **[layout]** est celui des exemples précédents :

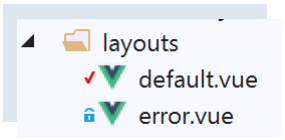
```
1.  <!-- disposition des vues -->
2.  <template>
3.    <!-- ligne -->
4.    <div>
5.      <b-row>
6.        <!-- zone à trois colonnes -->
7.        <b-col v-if="left" cols="3">
8.          <slot name="left" />
9.        </b-col>
10.       <!-- zone à neuf colonnes -->
11.       <b-col v-if="right" cols="9">
12.         <slot name="right" />
13.       </b-col>
14.     </b-row>
15.   </div>
16. </template>
17.
18. <script>
19. export default {
20.   // paramètres
21.   props: {
22.     left: {
23.       type: Boolean
24.     },
25.     right: {
26.       type: Boolean
27.     }
```

```
  }
}
</script>
```

Le composant **[navigation]** est le suivant :

```
<template>
  <!-- menu Bootstrap à trois options -->
  <b-nav vertical>
    <b-nav-item to="/authentification" exact exact-active-class="active">
      Authentification
    </b-nav-item>
    <b-nav-item to="/get-admindata" exact exact-active-class="active">
      Requête AdminData
    </b-nav-item>
    <b-nav-item to="/fin-session" exact exact-active-class="active">
      Fin session impôt
    </b-nav-item>
  </b-nav>
</template>
```

4.15.12 Les layouts de l'application [nuxt]



4.15.12.1 [default]

Le layout **[default]** est celui utilisé pour l'exemple **[nuxt-11]** au paragraphe lien :

```
<template>
  <div class="container">
    <b-card>
      <!-- un message -->
      <b-alert show variant="success" align="center">
        <h4>[nuxt-12] : requêtes HTTP avec axios</h4>
      </b-alert>
      <!-- la vue courante du routage -->
      <nuxt />
      <!-- message d'attente -->
      <b-alert v-if="showLoading" show variant="light">
        <strong>Requête au serveur de données en cours...</strong>
        <div class="spinner-border ml-auto" role="status" aria-hidden="true"></div>
      </b-alert>
      <!-- erreur d'une opération asynchrone -->
      <b-alert v-if="showErrorLoading" show variant="danger">
        <strong>La requête au serveur de données a échoué : {{ errorLoadingMessage }}</strong>
      </b-alert>
    </b-card>
  </div>
</template>

<script>
/* eslint-disable no-console */
export default {
  name: 'App',
  data() {
    return {
      showLoading: false,
      showErrorLoading: false
    }
  },
  // cycle de vie
  beforeCreate() {
    console.log('[default beforeCreate]')
  },
  created() {
    console.log('[default created]')
    if (process.client) {
```

```
40.       // on écoute l'évt [loading]
41.       this.$eventBus().$on('loading', this.mShowLoading)
42.       // ainsi que l'évt [errorLoading]
43.       this.$eventBus().$on('errorLoading', this.mShowErrorLoading)
44.     }
45.   },
46.   beforeMount() {
47.     console.log('[default beforeMount]')
48.   },
49.   mounted() {
50.     console.log('[default mounted]')
51.   },
52.   methods: {
53.     // gestion du message d'attente
54.     mShowLoading(value) {
55.       console.log('[default mShowLoading], showLoading=', value)
56.       this.showLoading = value
57.     },
58.     // erreur d'une opération asynchrone
59.     mShowErrorLoading(value, errorLoadingMessage) {
60.       console.log('[default mShowErrorLoading], showErrorLoading=', value, 'errorLoadingMessage=', errorLoa
    dingMessage)
61.       this.showErrorLoading = value
62.       this.errorLoadingMessage = errorLoadingMessage
63.     }
64.   }
65. }
66. </script>
```

- lignes 10-14 : affichent le message d'attente de la fin d'une opération asynchrone du client **[nuxt]** ;
- lignes 15-18 : affichent l'éventuel message d'erreur d'une opération asynchrone ;
- ligne 37 : la fonction **[created]** de la page **[default]** est exécutée avant la fonction **[mounted]** des pages ;
- ligne 39 : si l'exécuteur est le client **[nuxt]**, alors la page **[default]** se met à l'écoute des événements :
- **[loading]** qui signale le début ou la fin d'une attente. La fonction **[mShowLoading]** est alors exécutée ;
- **[errorLoading]** qui signale qu'il faut afficher un message d'erreur. La fonction **[mShowErrorLoading]** est alors exécutée ;
- les pages **[nuxt]** :
  - font afficher le message d'attente en émettant l'événement **['loading', true]** sur le bus d'événements ;
  - cachent le message d'attente en émettant l'événement **['loading', false]** sur le bus d'événements ;
  - font afficher un message d'erreur en émettant l'événement **['errorLoading', true]** sur le bus d'événements ;
  - cachent le message d'erreur en émettant l'événement **['errorLoading', false]** sur le bus d'événements ;

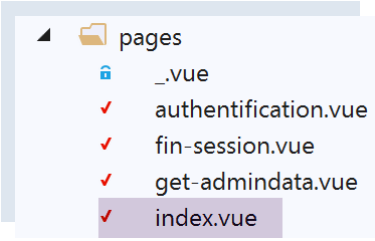
4.15.12.2 [error]

Le layout **[error]** affiche un message d'erreur système (non géré par le développeur) :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- mise en page -->
4.    <Layout :left="true" :right="true">
5.      <!-- alerte dans la colonne de droite -->
6.      <template slot="right">
7.        <!-- message sur fond rose -->
8.        <b-alert show variant="danger" align="center">
9.          <h4>L'erreur suivante s'est produite : {{ JSON.stringify(error) }}</h4>
10.       </b-alert>
11.     </template>
12.     <!-- menu de navigation dans la colonne de gauche -->
13.     <Navigation slot="left" />
14.   </Layout>
15. </template>
16.
17. <script>
18. /* eslint-disable no-undef */
19. /* eslint-disable no-console */
20. /* eslint-disable nuxt/no-env-in-hooks */
21.
22. import Layout from '@/components/layout'
23. import Navigation from '@/components/navigation'
24.
25. export default {
26.   name: 'Error',
27.   // composants utilisés
28.   components: {
29.     Layout,
```

```
    Navigation
  },
  // propriété [props]
  props: { error: { type: Object, default: () => 'waiting ...' } },
  // cycle de vie
  beforeCreate() {
    // client et serveur
    console.log('[error beforeCreate]')
  },
  created() {
    // client et serveur
    console.log('[error created, error=]', this.error)
  },
  beforeMount() {
    // client seulement
    console.log('[error beforeMount]')
  },
  mounted() {
    // client seulement
    console.log('[error mounted]')
  }
}
</script>
```

4.15.13 La page `[index]` exécutée par le serveur `[nuxt]`



La page **[index.vue]** a la particularité d'être accessible uniquement via le serveur **[nuxt]**. Aucun lien n'est présenté à l'utilisateur pour y avoir accès via le client **[nuxt]**. Son code est le suivant :

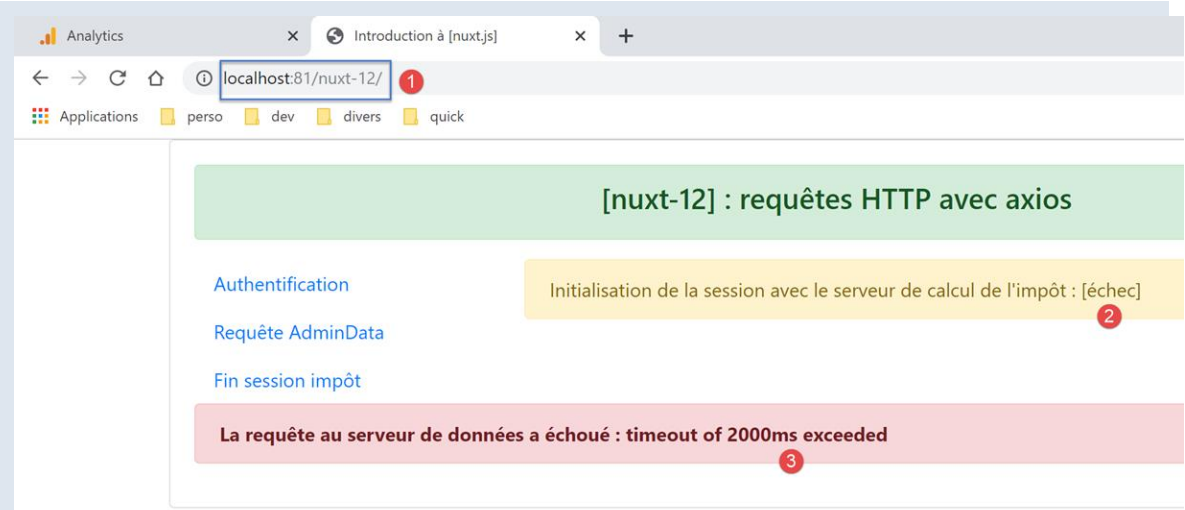
```
<!-- page principale -->
<template>
  <Layout :left="true" :right="true">
    <!-- navigation -->
    <Navigation slot="left" />
    <!-- message-->
    <b-
alert slot="right" show variant="warning">Initialisation de la session avec le serveur de calcul de l'impôt
 : {{ result }} </b-alert>
  </Layout>
</template>

<script>
/* eslint-disable no-console */

import Navigation from '@/components/navigation'
import Layout from '@/components/layout'

export default {
  name: 'InitSession',
  // composants utilisés
  components: {
    Layout,
    Navigation
  },
  // données asynchrones
  async asyncData(context) {
    // log
    console.log('[index asyncData started]')
    try {
      // on démarre une session jSON
      const dao = context.app.$dao()
      const response = await dao.initSession()
      // log
      console.log('[index asyncData response=]', response)
```

```
34.       // on récupère le cookie de session PHP pour les prochaines requêtes
35.       const phpSessionCookie = dao.getPhpSessionCookie()
36.       // on mémorise le cookie de session PHP dans la session [nuxt]
37.       context.store.commit('replace', { phpSessionCookie })
38.       // y-a-t-il eu erreur ?
39.       if (response.état !== 700) {
40.         // l'erreur se trouve dans response.réponse
41.         throw new Error(response.réponse)
42.       }
43.       // on note le fait que la session jSON a démarré
44.       context.store.commit('replace', { jsonSessionStarted: true })
45.       // on rend le résultat
46.       return { result: '[succès]' }
47.     } catch (e) {
48.       // log
49.       console.log('[index asyncData error=]', e)
50.       // on note le fait que la session jSON n'a pas démarré
51.       context.store.commit('replace', { jsonSessionStarted: false })
52.       // on signale l'erreur
53.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
54.     } finally {
55.       // on sauvegarde le store
56.       const session = context.app.$session()
57.       session.save(context)
58.       // log
59.       console.log('[index asyncData finished]')
60.     }
61.   },
62.   // cycle de vie
63.   beforeCreate() {
64.     console.log('[index beforeCreate]')
65.   },
66.   created() {
67.     console.log('[index created]')
68.   },
69.   beforeMount() {
70.     console.log('[index beforeMount]')
71.   },
72.   mounted() {
73.     console.log('[index mounted]')
74.     // client seulement
75.     if (this.showErrorLoading) {
76.       console.log('[index mounted, showErrorLoading=true]')
77.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
78.     }
79.   }
80. }
81. </script>
```

- ligne 7 : la page affiche le résultat **[result]** d'une requête asynchrone (lignes 46 et 51) ;
- ligne 31 : l'opération asynchrone est l'ouverture d'une session jSON avec le serveur de calcul de l'impôt ;
- ligne 25 : on sait que lorsque la page est demandée directement au serveur **[nuxt]**, la fonction **[asyncData]** n'est exécutée que par le serveur et pas par le client **[nuxt]** qui s'exécute lorsque le navigateur a reçu la réponse du serveur **[nuxt]** ;
- ligne 30 : on récupère la couche **[dao]** dans le contexte du serveur **[nuxt]** ;
- ligne 35 : si le serveur n'avait pas encore fait de requête au serveur de calcul de l'impôt, il reçoit son premier cookie de session PHP, sinon le dernier cookie de session PHP qu'il a reçu (revoir le code de la couche **[dao]** du serveur **[nuxt]** au paragraphe lien) ;
- ligne 37 : on mémorise ce cookie de session PHP dans le store ;
- lignes 39-42 : on regarde si l'opération a réussi. Si ce n'est pas le cas, une exception est lancée qui sera interceptée par le **[catch]** de la ligne 47 ;
- ligne 44 : on note dans le store que la session jSON avec le serveur PHP est démarrée ;
- ligne 46 : on rend le résultat **[result]** qui est affiché ligne 7 ;
- lignes 47-54 : on traite une éventuelle exception. Celle-ci peut-être de deux natures :
- l'opération HTTP de la ligne 31 a échoué sur une erreur de communication serveur **[nuxt]** / serveur PHP ;
- l'opération HTTP de la ligne 31 a réussi mais le résultat reçu a signalé une erreur (lignes 39-42) ;
- ligne 51 : on note que la session jSON avec le serveur PHP n'a pas démarré ;
- ligne 53 : on rend le résultat **[result]** qui est affiché ligne 7. Par ailleurs, on positionne les propriétés **[showErrorLoading]** et **[errorLoadingMessage]** que le client **[nuxt]** va utiliser pour afficher un message d'erreur lorsqu'il recevra la page envoyée par le serveur **[nuxt]** (lignes 72-79) ;
- lignes 54-60 : code exécuté dans tous les cas (réussite ou échec) ;
- ligne 56 : on récupère la session **[nuxt]** dans le contexte du serveur **[nuxt]** ;
- ligne 57 : on la sauvegarde ;

- lignes 63-68 : une fois la fonction **[asyncData]** terminée, le serveur **[nuxt]** exécute les fonctions **[beforeCreate]** et **[create]** ;

Note : l'exécution de la page **[index]** par le serveur **[nuxt]** peut échouer par exemple si le serveur de calcul de l'impôt n'est pas lancé lorsque l'application **[nuxt]** est elle lancée :



Dans ce cas, la seule solution est de lancer le serveur de calcul de l'impôt puis l'application **[nuxt]** elle-même puisque le menu de navigation ne propose pas d'option pour initier une session jSON avec le serveur de calcul de l'impôt ;

4.15.14 La page [index] exécutée par le client [nuxt]

La page **[index]** n'est exécutée par le client **[nuxt]** qu'après que le serveur **[nuxt]** la lui ait envoyée. Celui-ci lui a envoyé les informations **[result]** et éventuellement **[showErrorLoading]** et **[errorLoadingMessage]**.

On sait que la fonction **[asyncData]** ne sera pas exécutée. Restent alors les fonctions du cycle de vie et notamment la fonction **[mounted]** :

```
1.    mounted() {
2.      console.log('[index mounted]')
3.      // client seulement
4.      if (this.showErrorLoading) {
5.        console.log('[index mounted, showErrorLoading=true]')
6.        this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
7.      }
8.    }
```

- le client **[nuxt]** intègre automatiquement dans les propriétés de la page les éléments **[result]** et éventuellement **[showErrorLoading, errorLoadingMessage]** que lui a envoyés le serveur **[nuxt]** :
- la propriété **[result]** est affichée par la ligne 7 ;
- les propriétés **[showErrorLoading, errorLoadingMessage]** sont utilisées par la méthode **[mounted]** : ligne 4, on teste la propriété **[showErrorLoading]**. Si elle est vraie, on utilise, ligne 6, le bus d'événements du client **[nuxt]** pour signaler qu'il y a un message d'erreur à afficher ;
- l'événement **[errorLoading]** lancé ligne 6, est intercepté par la page **[layouts/default]** décrite au paragraphe lien ;

4.15.15 La page [authentification] exécutée par le serveur [nuxt]

La page **[authentification]** est chargée d'identifier un utilisateur auprès du serveur de calcul de l'impôt. Son code est le suivant :

```
1.  <!-- page d'authentification -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-
    alert slot="right" show variant="warning">Authentification auprès du serveur de calcul de l'impôt : {{ result }} </b-alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
```

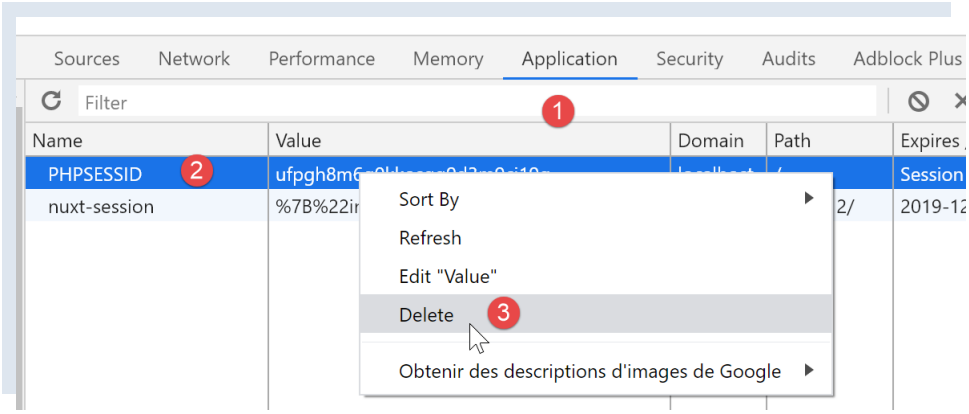
```
13.
14.  import Navigation from '@/components/navigation'
15.  import Layout from '@/components/layout'
16.
17.  export default {
18.    name: 'Authentification',
19.    // composants utilisés
20.    components: {
21.      Layout,
22.      Navigation
23.    },
24.    // données asynchrones
25.    async asyncData(context) {
26.      // log
27.      console.log('[authentification asyncData started]')
28.      if (process.client) {
29.        // début attente
30.        context.app.$eventBus().$emit('loading', true)
31.        // pas d'erreur
32.        context.app.$eventBus().$emit('errorLoading', false)
33.      }
34.      try {
35.        // on s'authentifie auprès du serveur
36.        const dao = context.app.$dao()
37.        const response = await dao.authentifierUtilisateur('admin', 'admin')
38.        // log
39.        console.log('[authentification asyncData response=]', response)
40.        // résultat
41.        const userAuthenticated = response.état === 200
42.        // on note dans le store le fait que l'utilisateur est authentifié ou pas
43.        context.store.commit('replace', { userAuthenticated })
44.        // on sauvegarde le store dans la session [nuxt]
45.        const session = context.app.$session()
46.        session.save(context)
47.        // erreur d'authentification ?
48.        if (!userAuthenticated) {
49.          // l'erreur se trouve dans response.réponse
50.          throw new Error(response.réponse)
51.        }
52.        // on rend le résultat
53.        return { result: '[succès]' }
54.      } catch (e) {
55.        // on signale l'erreur
56.        return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
57.      } finally {
58.        // log
59.        console.log('[authentification asyncData finished]')
60.        if (process.client) {
61.          // fin attente
62.          context.app.$eventBus().$emit('loading', false)
63.        }
64.      }
65.    },
66.    // cycle de vie
67.    beforeCreate() {
68.      console.log('[authentification beforeCreate]')
69.    },
70.    created() {
71.      console.log('[authentification created]')
72.    },
73.    beforeMount() {
74.      console.log('[authentification beforeMount]')
75.    },
76.    mounted() {
77.      console.log('[authentification mounted]')
78.      // client seulement
79.      if (this.showErrorLoading) {
80.        console.log('[authentification mounted, showErrorLoading=true]')
81.        this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
82.      }
83.    }
84.  }
85.  </script>
```

- ligne 7 : la page affiche le résultat **[result]** de la requête asynchrone **[asyncData]** des lignes 25-65 ;
- lignes 28-33 : le serveur n'exécute pas ces lignes destinées au client **[nuxt]** ;
- ligne 36 : on récupère la couche **[dao]** du serveur **[nuxt]** ;
- ligne 37 : on s'authentifie auprès du serveur de calcul de l'impôt avec les identifiants de test **[admin, admin]** qui sont les seuls acceptés par le serveur de calcul de l'impôt ;
- ligne 41 : l'opération d'authentification a réussi si seulement la réponse à l'état 200 ;
- ligne 43 : on met dans le store la propriété **[userAuthenticated]** ;
- lignes 44-46 : le store est sauvegardé dans la session **[nuxt]** ;

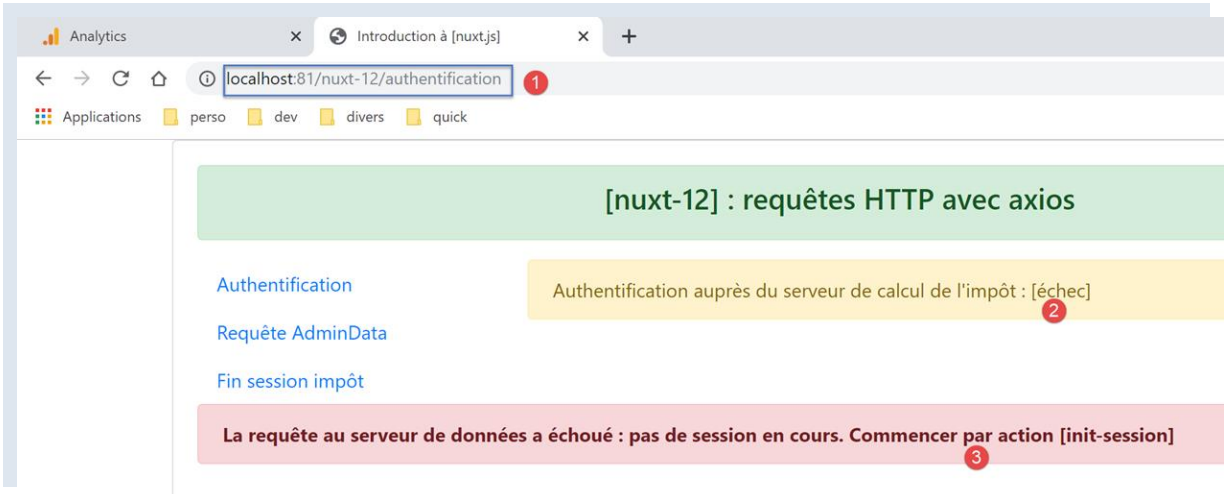
- lignes 48-51 : si l'authentification a échoué, on lance une exception avec le message d'erreur que le serveur de calcul de l'impôt a envoyé ;
- sinon ligne 53, on retoune un résultat de réussite qui sera affiché ligne 7 ;
- lignes 54-57 : en cas d'erreur on positionne trois propriétés de la page **[result, showErrorLoading, errorLoadingMessage]**. La propriété **[result]** sera affichée ligne 7. Les trois propriétés seront envoyées au client **[nuxt]** ;
- lignes 60-63 : ne sont pas exécutées par le serveur **[nuxt]** ;
- une fois que **[asyncData]** a rendu son résultat, celui-ci est affiché ligne 7. Puis les méthodes **[beforeCreate]** (lignes 67-69) et **[created]** (lignes 70-72) sont exécutées ;
- c'est fini ;

Note : l'exécution de la page **[authentification]** par le serveur **[nuxt]** peut échouer par exemple si la session jSON avec le serveur de calcul de l'impôt n'a pas été initialisée. Cela est possible de la façon suivante :

- supprimez le cookie de session PHP de votre navigateur (pour repartir de zéro) :



- lancez l'application **[nuxt]** alors que le serveur de calcul n'a pas été lancé : vous obtenez une erreur ;
- lancez le serveur de calcul de l'impôt ;
- demandez l'URL **[/authentification]** directement dans la barre d'adresses du navigateur :



Dans ce cas, la seule solution est de nouveau de recharger la page **[index]**.

4.15.16 La page [authentification] exécutée par le client [nuxt]

Reprenons le code de la page :

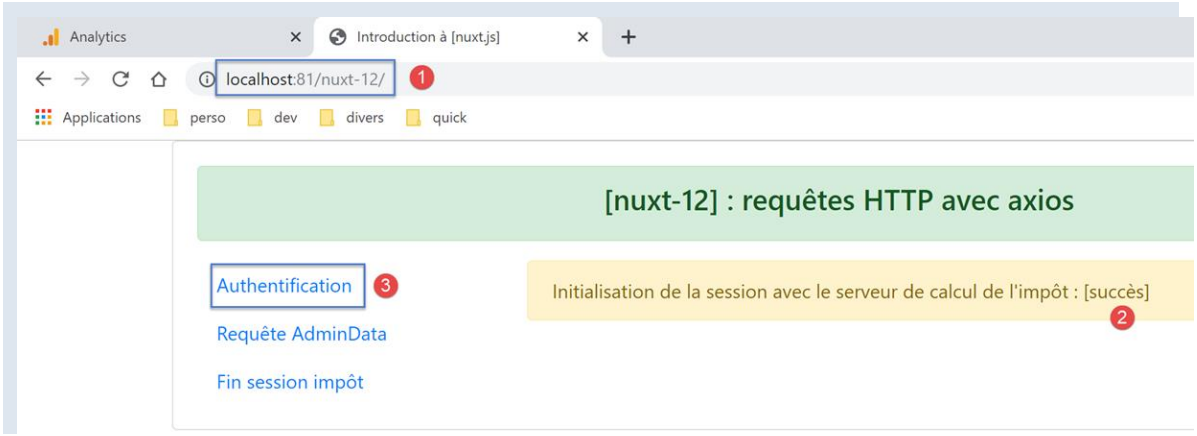
```
1.  <!-- page d'authentification -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
```

```
      <b-alert slot="right" show variant="warning">Authentification auprès du serveur de calcul de l'impôt : {{ result }} </b-alert>
    </Layout>
  </template>

  <script>
  /* eslint-disable no-console */

  import Navigation from '@/components/navigation'
  import Layout from '@/components/layout'

  export default {
    name: 'Authentification',
    // composants utilisés
    components: {
      Layout,
      Navigation
    },
    // données asynchrones
    async asyncData(context) {
      // log
      console.log('[authentification asyncData started]')
      if (process.client) {
        // début attente
        context.app.$eventBus().$emit('loading', true)
        // pas d'erreur
        context.app.$eventBus().$emit('errorLoading', false)
      }
      try {
        // on s'authentifie auprès du serveur
        const dao = context.app.$dao()
        const response = await dao.authentifierUtilisateur('admin', 'admin')
        // log
        console.log('[authentification asyncData response=]', response)
        // résultat
        const userAuthenticated = response.état === 200
        // on note dans le store le fait que l'utilisateur est authentifié ou pas
        context.store.commit('replace', { userAuthenticated })
        // on sauvegarde le store dans la session [nuxt]
        const session = context.app.$session()
        session.save(context)
        // erreur d'authentification ?
        if (!userAuthenticated) {
          // l'erreur se trouve dans response.réponse
          throw new Error(response.réponse)
        }
        // on rend le résultat
        return { result: '[succès]' }
      } catch (e) {
        // on signale l'erreur
        return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
      } finally {
        // log
        console.log('[authentification asyncData finished]')
        if (process.client) {
          // fin attente
          context.app.$eventBus().$emit('loading', false)
        }
      }
    },
    // cycle de vie
    beforeCreate() {
      console.log('[authentification beforeCreate]')
    },
    created() {
      console.log('[authentification created]')
    },
    beforeMount() {
      console.log('[authentification beforeMount]')
    },
    mounted() {
      console.log('[authentification mounted]')
      // client seulement
      if (this.showErrorLoading) {
        console.log('[authentification mounted, showErrorLoading=true]')
        this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
```

```
82.     }
83.   }
84. }
85. </script>
```

Il y a deux cas d'exécution de la page **[authentification]** par le client **[nuxt]** :

- le client **[nuxt]** s'exécute après que le serveur **[nuxt]** ait envoyé au navigateur du client **[nuxt]** la page **[authentification]** ;
- le client **[nuxt]** parce que l'utilisateur a cliqué sur le lien **[Authentification]** du menu de navigation :



Etudions d'abord le 1er cas. Dans ce cas, le client **[nuxt]** n'exécute pas la fonction **[asyncData]**. Il intègre dans les propriétés de la page les éléments **[result]** et éventuellement **[showErrorLoading, errorLoadingMessage]** que lui a envoyés le serveur **[nuxt]** :

- la propriété **[result]** est affichée par la ligne 7 ;
- les propriétés **[showErrorLoading, errorLoadingMessage]** sont utilisées par la méthode **[mounted]** : ligne 79, on teste la propriété **[showErrorLoading]**. Si elle est vraie, on utilise, ligne 81, le bus d'événements du client **[nuxt]** pour signaler qu'il y a un message d'erreur à afficher ;

Le mécanisme de l'affichage du message d'erreur a été expliqué pour la page **[index]** au paragraphe lien.

Le cas 2 est celui du client **[nuxt]** exécuté lorsque l'utilisateur clique sur le lien **[Authentification]**. Dans ce cas, le client **[nuxt]** s'exécute de façon autonome et pas après le serveur **[nuxt]**. La fonction **[asyncData]** est alors exécutée. Nous ne donnons que les détails qui diffèrent des explications données pour la page exécutée par le serveur **[nuxt]** :

- lignes 28-33 : le client **[nuxt]** demande l'affichage du message d'attente et la disparition d'un éventuel message d'erreur qui aurait été précédemment affiché ;
- ligne 36 : c'est désormais la couche **[dao]** du client **[nuxt]** qui est obtenue ici ;
- lignes 60-63 : le client **[nuxt]** demande la fin de l'affichage du message d'attente ;
- une fois **[asyncData]** terminée, le cycle de vie de la page va avoir lieu. la fonction **[mounted]** des lignes 76-83 va être exécutée. S'il y a eu erreur, le message d'erreur va alors être affiché ;

Note : pour provoquer une erreur, suivez la procédure expliquée pour le serveur **[nuxt]** à la fin du paragraphe lien, mais au lieu de demander la page **[authentification]** en tapant son URL dans la barre d'adresses, utilisez le lien **[Authentification]** du menu de navigation. C'est alors le client **[nuxt]** qui s'exécute.

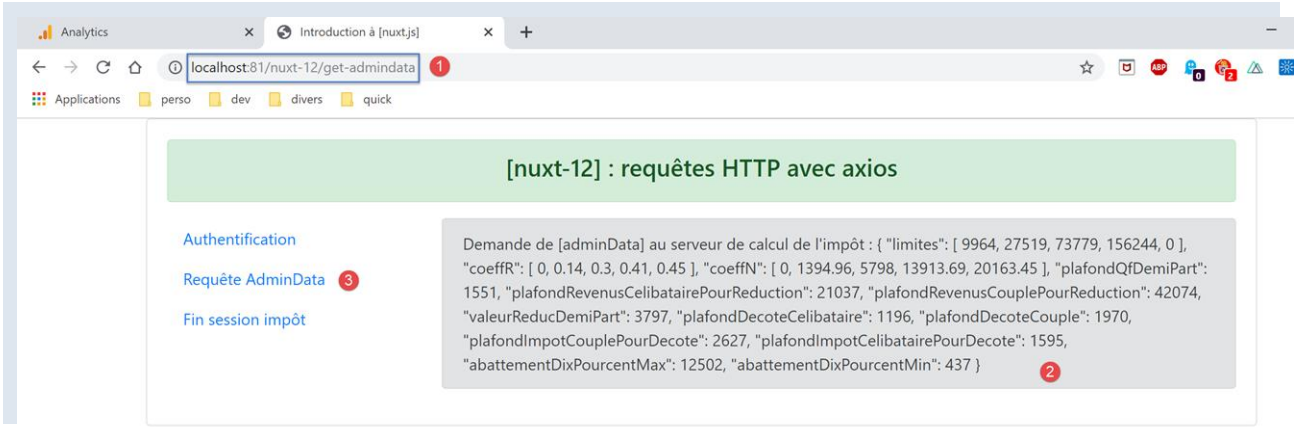
4.15.17 La page [get-admindata]

Le code de la page **[get-admindata]** est le suivant :

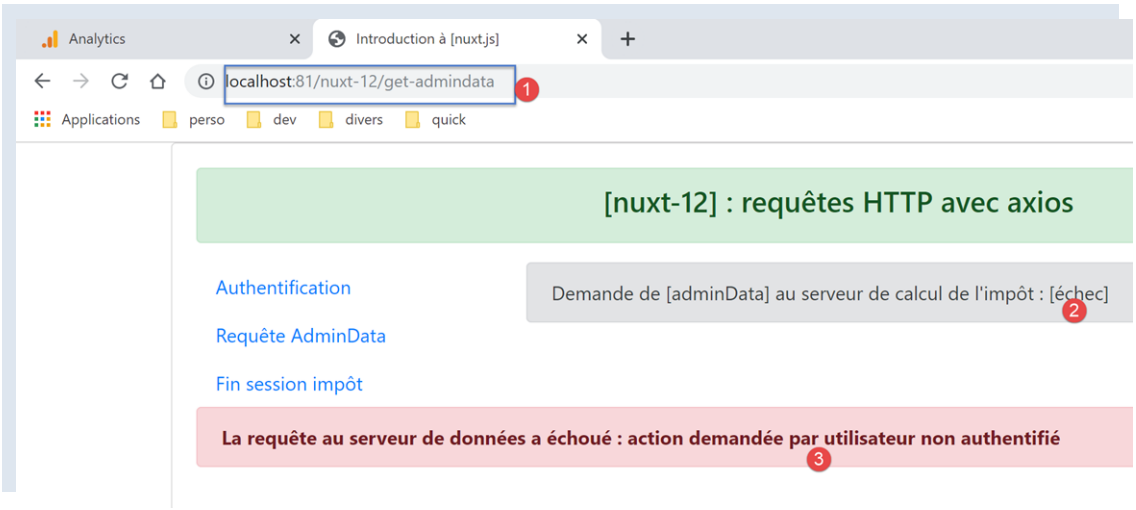
```
1.  <!-- vue get-admindata -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message -->
7.      <b-
    alert slot="right" show variant="secondary"> Demande de [adminData] au serveur de calcul de l'impôt : {{ re
    sult }} </b-alert>
8.    </Layout>
9.  </template>
```

```
10.
11. <script>
12. /* eslint-disable no-console */
13.
14. import Navigation from '@/components/navigation'
15. import Layout from '@/components/layout'
16.
17. export default {
18.   name: 'GetAdmindata',
19.   // composants utilisés
20.   components: {
21.     Layout,
22.     Navigation
23.   },
24.   // données asynchrones
25.   async asyncData(context) {
26.     // log
27.     console.log('[get-admindata asyncData started]')
28.     if (process.client) {
29.       // début attente
30.       context.app.$eventBus().$emit('loading', true)
31.       // pas d'erreur
32.       context.app.$eventBus().$emit('errorLoading', false)
33.     }
34.     try {
35.       // on demande la donnée [admindata]
36.       const response = await context.app.$dao().getAdminData()
37.       // log
38.       console.log('[get-admindata asyncData response=]', response)
39.       // résultat
40.       const adminData = response.état === 1000 ? response.réponse : ''
41.       // on met la donnée dans le store
42.       context.store.commit('replace', { adminData })
43.       // on sauvegarde le store dans la session [nuxt]
44.       const session = context.app.$session()
45.       session.save(context)
46.       // y-a-t-il eu erreur ?
47.       if (!adminData) {
48.         // l'erreur se trouve dans response.réponse
49.         throw new Error(response.réponse)
50.       }
51.       // on rend la valeur reçue
52.       return { result: adminData }
53.     } catch (e) {
54.       // on signale l'erreur
55.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
56.     } finally {
57.       // log
58.       console.log('[get-admindata asyncData finished]')
59.       if (process.client) {
60.         // fin attente
61.         context.app.$eventBus().$emit('loading', false)
62.       }
63.     }
64.   },
65.   // cycle de vie
66.   beforeCreate() {
67.     console.log('[get-admindata beforeCreate]')
68.   },
69.   created() {
70.     console.log('[get-admindata created]')
71.   },
72.   beforeMount() {
73.     console.log('[get-admindata beforeMount]')
74.   },
75.   mounted() {
76.     console.log('[get-admindata mounted]')
77.     // client
78.     if (this.showErrorLoading) {
79.       console.log('[get-admindata mounted, showErrorLoading=true]')
80.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
81.     }
82.   }
83. }
84. </script>
```

Cette page est très semblable à la page **[authentification]**. Les explications sont analogues aussi bien pour son exécution par le serveur **[nuxt]** que pour son exécution par le client **[nuxt]**. Notons cependant que la ligne 7 affiche non pas succès / échec comme précédemment mais la valeur de la donnée reçue du serveur de calcul de l'impôt (ligne 52) :



Le résultat ci-dessus est obtenu aussi bien avec le serveur qu'avec le client **[nuxt]**. Pour provoquer une erreur, demandez la page **[get-admindata]**, via le serveur ou le client **[nuxt]**, sans être authentifié :



4.15.18 La page [fin-session]

Le code de la page est le suivant :

```
1.  <!-- page principale -->
2.  <template>
3.    <Layout :left="true" :right="true">
4.      <!-- navigation -->
5.      <Navigation slot="left" />
6.      <!-- message-->
7.      <b-alert slot="right" show variant="warning">Fin de la session avec le serveur de calcul de l'impôt : {{ result }} </b-
    alert>
8.    </Layout>
9.  </template>
10.
11. <script>
12. /* eslint-disable no-console */
13.
14. import Navigation from '@/components/navigation'
15. import Layout from '@/components/layout'
16.
17. export default {
18.   name: 'FinSession',
19.   // composants utilisés
20.   components: {
21.     Layout,
22.     Navigation
23.   },
24.   // données asynchrones
```

```
25.    async asyncData(context) {
26.      // log
27.      console.log('[fin-session asyncData started]')
28.      // cas du client [nuxt]
29.      if (process.client) {
30.        // début attente
31.        context.app.$eventBus().$emit('loading', true)
32.        // pas d'erreur
33.        context.app.$eventBus().$emit('errorLoading', false)
34.      }
35.      try {
36.        // on demande une nouvelle session PHP au serveur de calcul de l'impôt
37.        const dao = context.app.$dao()
38.        const response = await dao.finSession()
39.        // log
40.        console.log('[fin-session asyncData response=]', response)
41.        // y-at-il eu erreur ?
42.        if (response.état !== 400) {
43.          // l'erreur se trouve dans response.réponse
44.          throw new Error(response.réponse)
45.        }
46.        // le serveur a envoyé un nouveau cookie de session PHP
47.        // on le récupère à la fois pour le serveur et le client nuxt
48.        // si ce code est exécuté par le client, le cookie de session PHP doit être mis dans la session nuxt
49.        // pour que le plugin [plgDao] puisse le récupérer et initialiser la couche [dao] avec
50.        // si ce code est exécuté par le serveur, le cookie de session PHP doit être mis dans la session nuxt
51.        // pour que le routing du client le récupère et le passe au navigateur
52.
53.        const phpSessionCookie = dao.getPhpSessionCookie()
54.        // on note dans le store le fait que la session jSON est démarrée et on mémorise le cookie de session PHP
55.        context.store.commit('replace', { jsonSessionStarted: true, phpSessionCookie, userAuthenticated: false, adminData: ''
     })
56.        // on sauvegarde le store dans la session [nuxt]
57.        const session = context.app.$session()
58.        session.save(context)
59.        // on rend le résultat
60.        return { result: "[succès]. La session jSON reste initialisée mais vous n'êtes plus authentifié(e)." }
61.      } catch (e) {
62.        // log
63.        console.log('[fin-session asyncData error=]', e)
64.        // on signale l'erreur
65.        return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
66.      } finally {
67.        // log
68.        console.log('[fin-session asyncData finished]')
69.        if (process.client) {
70.          // fin attente
71.          context.app.$eventBus().$emit('loading', false)
72.        }
73.      }
74.    },
75.    // cycle de vie
76.    beforeCreate() {
77.      console.log('[fin-session beforeCreate]')
78.    },
79.    created() {
80.      console.log('[fin-session created]')
81.    },
82.    beforeMount() {
83.      console.log('[fin-session beforeMount]')
84.    },
85.    mounted() {
86.      console.log('[fin-session mounted]')
87.      // client seulement
88.      if (this.showErrorLoading) {
89.        console.log('[fin-session mounted, showErrorLoading=true]')
90.        this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
91.      }
92.    }
93.  }
94.  </script>
```

Le code est très analogue à celui des pages précédentes et les explications sont les mêmes. Il faut simplement s'attarder sur un point : l'opération asynchrone de la ligne 38, fait que le serveur de calcul de l'impôt va envoyer un nouveau cookie de session PHP. Les explications pour la gestion de ce cookie diffèrent selon que c'est le serveur ou le client **[nuxt]** qui exécute ce code.

Commençons par le serveur **[nuxt]** :

- ligne 37 : c'est la couche **[dao]** du serveur **[nuxt]** qui est instanciée. Rappelons le code de son constructeur :

```
// constructeur
  constructor(axios, phpSessionCookie) {
```

```
    // bibliothèque axios
    this.axios = axios
    // valeur du cookie de session
    this.phpSessionCookie = phpSessionCookie
    // nom du cookie de session du serveur PHP
    this.phpSessionCookieName = 'PHPSESSID'
}
```

On voit ligne 1, que le constructeur a besoin du cookie de session PHP du moment, le dernier reçu, que ce soit par le serveur ou le client **[nuxt]** ;

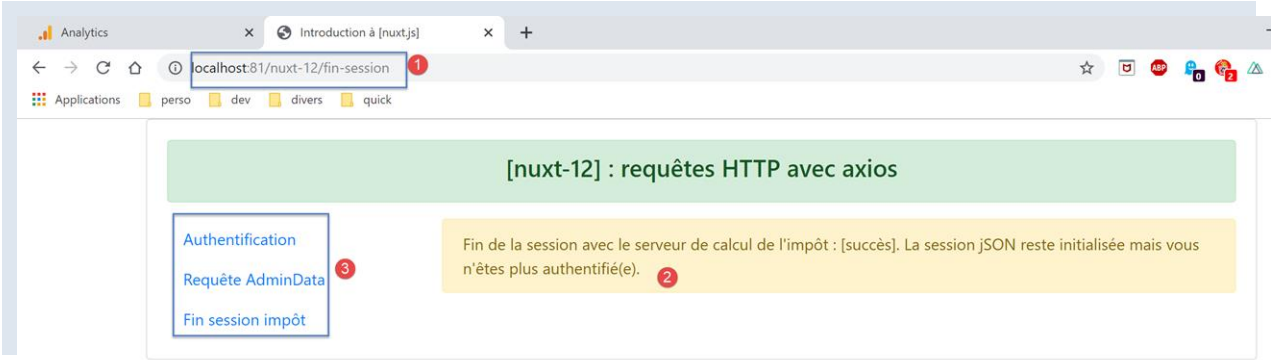
- ligne 52 : le serveur **[nuxt]** récupère le cookie de la nouvelle session PHP ou bien l'ancien cookie si l'opération de fin de session a échoué ;
- ligne 54 : le cookie de session PHP est mis dans le store puis sauvegardé dans la session **[nuxt]** aux lignes 56-57 ;
- après le serveur c'est le client **[nuxt]** qui exécute la page **[fin-session]** avec les données envoyées par le serveur. On sait qu'il ne va pas exécuter la fonction **[asyncData]** ;
- au final, après que serveur et client **[nuxt]** ont terminé leur travail, on sait que le cookie PHP nécessaire aux échanges avec le serveur de calcul de l'impôt est dans la session **[nuxt]** ;

Le fait que le cookie PHP soit dans la session **[nuxt]** est suffisant pour le serveur, car c'est là que va le prendre sa couche **[dao]**. Dans le plugin **[server/plgDao]** qui initialise la couche **[dao]** du serveur, on a écrit :

```
1.  /* eslint-disable no-console */
2.  // on crée un point d'accès à la couche [Dao]
3.  import Dao from '@/api/server/Dao'
4.  export default (context, inject) => {
5.    // configuration axios
6.    context.$axios.defaults.timeout = context.env.timeout
7.    context.$axios.defaults.baseURL = context.env.baseURL
8.    // on récupère le cookie de session
9.    const store = context.app.$session().value.store
10.   const phpSessionCookie = store ? store.phpSessionCookie : ''
11.   console.log('session=', context.app.$session().value, 'phpSessionCookie=', phpSessionCookie)
12.   // instanciation de la couche [dao]
13.   const dao = new Dao(context.$axios, phpSessionCookie)
14.   // injection d'une fonction [$dao] dans le contexte
15.   inject('dao', () => dao)
16.   // log
17.   console.log('[fonction server $dao créée]')
18. }
```

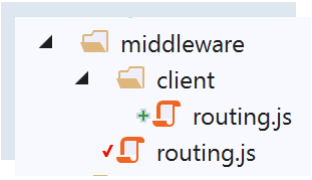
- ligne 13, la couche **[dao]** du serveur **[nuxt]** est instanciée avec le cookie de session PHP pris dans la session **[nuxt]**, lignes 9-10 ;

Pour le client **[nuxt]**, c'est une autre histoire. Ce n'est pas lui en effet qui envoie le cookie mais le navigateur qui l'exécute. Or ce navigateur ne connaît pas le cookie de la nouvelle session PHP reçu par le serveur **[nuxt]**. Si on utilise les liens du menu de navigation **[3]** :



Le serveur de calcul de l'impôt va recevoir du navigateur un cookie de session PHP obsolète et il va répondre qu'à ce cookie aucune session jSON n'est associée. Il nous faut trouver le moyen de passer au navigateur le nouveau cookie de session PHP.

On peut utiliser un middleware de routing pour ce faire :



Le script **[client/routing]** est le middleware de routage déclaré dans le fichier **[nuxt.config]** :

```
1.    // routeur
2.    router: {
3.      // racine des URL de l'application
4.      base: '/nuxt-12/',
5.      // middleware de routage
6.      middleware: ['routing']
7.    },
```

Le script **[middleware/routing]** est le suivant :

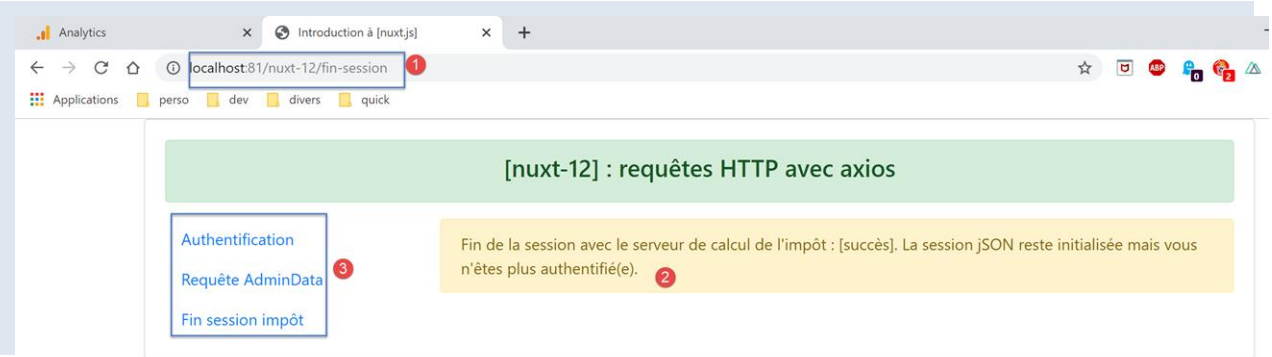
```
1.  /* eslint-disable no-console */
2.
3.  // on importe le middleware du client
4.  import clientRouting from './client/routing'
5.
6.  export default function(context) {
7.    // qui exécute ce code ?
8.    console.log('[middleware], process.server', process.server, ', process.client=', process.client)
9.    if (process.client) {
10.     // routage client
11.     clientRouting(context)
12.   }
13. }
```

- lignes 9-12 : on ne route que le client avec une fonction importée ligne 4 ;

Le script **[middleware/client/routing]** est le suivant :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware client], process.server', process.server, ', process.client=', process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session nuxt
7.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18. }
```

Revenons à la situation juste après l'exécution de la page **[fin-session]** par le serveur **[nuxt]** :



Si on clique sur l'un des liens du menu **[3]**, le client **[nuxt]** va prendre la main. Comme il va y avoir changement de page, le script de routing du client va s'exécuter :

- ligne 13 : le cookie de session PHP est trouvé dans le store de l'application **[nuxt]** ;
- ligne 14 : s'il n'est pas vide on le transmet au navigateur (ligne 16). A partir de ce moment le navigateur du client **[nuxt]** a le bon cookie de session PHP ;

Le script **[client/routing]** est exécuté à chaque changement de page du client **[nuxt]**. Le code du script est valide quelque soit la page cible : simplement, la plupart du temps, il donne au navigateur un cookie de session PHP qu'il a déjà, sauf dans deux cas :

- juste après le démarrage de l'application, le serveur **[nuxt]** exécute la page **[index]** et reçoit un 1er cookie de session PHP que le navigateur du client **[nuxt]** n'a pas ;
- lorsque le serveur **[nuxt]** exécute la page **[fin-session]** comme il vient d'être expliqué ;

Maintenant étudions le cas où la page **[fin-session]** est exécutée par le client **[nuxt]** uniquement, parce qu'on a cliqué sur son lien dans le menu de navigation. C'est désormais le client **[nuxt]** qui exécute la fonction **[asyncData]** :

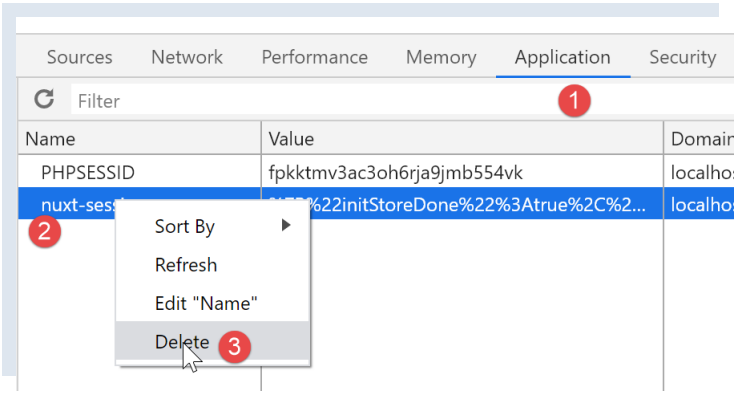
```
1.      try {
2.        // on demande une nouvelle session PHP au serveur de calcul de l'impôt
3.        const dao = context.app.$dao()
4.        const response = await dao.finSession()
5.        // log
6.        console.log('[fin-session asyncData response=]', response)
7.        // y-at-il eu erreur ?
8.        if (response.état !== 400) {
9.          // l'erreur se trouve dans response.réponse
10.         throw new Error(response.réponse)
11.       }
12.       // le serveur a envoyé un nouveau cookie de session PHP
13.       // on le récupère à la fois pour le serveur et le client nuxt
14.       // si ce code est exécuté par le client, le cookie de session PHP doit être mis dans la session nuxt
15.       // pour que le plugin [plgDao] puisse le récupérer et initialiser la couche [dao] avec
16.       // si ce code est exécuté par le serveur, le cookie de session PHP doit être mis dans la session nuxt
17.       // pour que le routing du client le récupère et le passe au navigateur
18.       const phpSessionCookie = dao.getPhpSessionCookie()
19.       // on note dans le store le fait que la session jSON est démarrée et on mémorise le cookie de session PHP
20.       context.store.commit('replace', { jsonSessionStarted: true, phpSessionCookie, userAuthenticated: false, adminData: ''
    })
21.       // on sauvegarde le store dans la session [nuxt]
22.       const session = context.app.$session()
23.       session.save(context)
24.       // on rend le résultat
25.       return { result: "[succès]. La session jSON reste initialisée mais vous n'êtes plus authentifié(e)." }
26.     } catch (e) {
27.       // log
28.       console.log('[fin-session asyncData error=]', e)
29.       // on signale l'erreur
30.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
31.     } finally {
32.       // log
33.       console.log('[fin-session asyncData finished]')
34.       if (process.client) {
35.         // fin attente
36.         context.app.$eventBus().$emit('loading', false)
37.       }
38.     }
```

- ligne 3 : c'est la couche **[dao]** du client **[nuxt]** qui est obtenu ici ;
- ligne 18 : le cookie de session PHP récupéré par la couche **[dao]** du client **[nuxt]** est mémorisé, mis dans le store (ligne 20) puis sauvegardé en session **[nuxt]** (lignes 22-23) ;

- à partir de là tout va bien car on sait que la couche **[dao]** du serveur **[nuxt]** va chercher le cookie de session PHP dans la session **[nuxt]** ;

4.15.19 Exécution

Pour exécuter cet exemple, il faut prendre soin avant l'exécution de supprimer le cookie de session **[nuxt]** et le cookie PHP du navigateur exécutant le client **[nuxt]** afin de partir d'une situation nette. Ci-dessous un exemple avec le navigateur Chrome :



4.15.20 Conclusion

Cet exemple a été particulièrement complexe. Il réunissait des connaissances acquises dans les exemples précédents : persistance du store dans une session **[nuxt]**, plugins d'injections de fonctions, middleware de routage, gestion des erreurs des opérations asynchrones. La complexité a été accrue par le fait qu'on voulait que l'utilisateur puisse aussi bien utiliser les liens du menu de navigation que taper des URL à la main sans que ça casse l'application. Pour cela, on a été obligés de regarder comment se comportait chaque page selon qu'elle était exécutée par le client ou le serveur **[nuxt]**.

Cette unité de comportement du client et du serveur **[nuxt]** n'est pas indispensable. On peut se mettre dans le cas fréquent où :

- la première page est délivrée par le serveur **[nuxt]** ;
- toutes les pages suivantes sont délivrées par le client **[nuxt]** qui travaille alors en mode **[SPA]** ;

Néanmoins même dans ce cas, il faut vérifier ce que donne l'exécution de toutes les pages par le serveur **[nuxt]** car c'est ce qu'obtiendront les moteurs de recherche qui les demanderont.

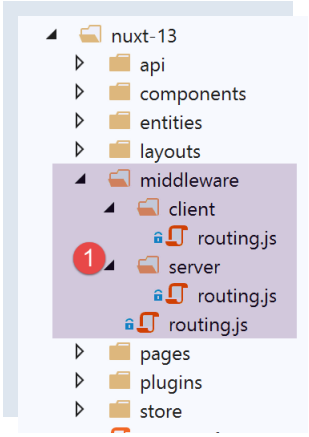
4.16 Exemple [nuxt-13] : contrôle de la navigation de [nuxt-12]

Dans cet exemple, nous nous intéressons à la navigation de **[nuxt-12]**. On ne l'a pas fait dans **[nuxt-12]** parce que le contrôle de la navigation aurait complexifié un exemple déjà complexe.

Objectif : nous voulons que l'utilisateur ne puisse faire que des actions autorisées :

- si la session jSON n'a pas démarré, alors seule l'URL **[/]** est autorisée ;
- si la session jSON a démarré mais que l'utilisateur n'est pas authentifié, alors seule l'URL **[/authentification]** est autorisée ;
- si la session jSON a démarré et que l'utilisateur est authentifié, alors seules les URL **[/get-admindata, /fin-session]** sont autorisées ;
- lorsque la cible du routage du moment n'est pas autorisée, alors on procèdera à une redirection vers une URL autorisée ;

L'exemple **[nuxt-13]** est obtenu initialement par recopie de l'exemple **[nuxt-12]** :



C'est dans le dossier du routage **[middleware]** que vont prendre place les modifications.

4.16.1 Routage de l'application [nuxt]

Le routage de l'application est configuré de la façon suivante dans le fichier **[nuxt.config]** :

```
1.    // routeur
2.    router: {
3.      // racine des URL de l'application
4.      base: '/nuxt-13/',
5.      // middleware de routage
6.      middleware: ['routing']
7.    },
```

- ligne 6 : le routage de l'application est contrôlé par le fichier **[middleware/routing]** ;

Le fichier **[middleware/routing]** est le suivant :

```
1.  /* eslint-disable no-console */
2.
3.  // on importe les middleware du serveur et du client
4.  import serverRouting from './server/routing'
5.  import clientRouting from './client/routing'
6.
7.  export default function(context) {
8.    // qui exécute ce code ?
9.    console.log('[middleware], process.server', process.server, ', process.client=', process.client)
10.   if (process.server) {
11.     // routage serveur
12.     serverRouting(context)
13.   } else {
14.     // routage client
15.     clientRouting(context)
16.   }
17. }
```

- lignes 10-16 : on traite différemment les routages du client et du serveur **[nuxt]**. C'est un point où ils diffèrent grandement ;
- ligne 4 : le routage du serveur est implémenté par le script **[middleware/server/routing]** ;
- ligne 5 : le routage du client est implémenté par le script **[middleware/client/routing]** ;

4.16.2 Routage du client [nuxt]

Le routage du client **[nuxt]** reste ce qu'il était dans **[nuxt-12]** :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware client], process.server', process.server, ', process.client=', process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session nuxt
7.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
```

```
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18.
19.   …
20. }
```

Pour éviter que le client aille dans des routes non autorisées on va simplement lui offrir dans le menu de navigation client les seules routes autorisées. Le composant **[components/navigation]** devient le suivant :

```
1.  <template>
2.    <!-- menu Bootstrap à trois options -->
3.    <b-nav vertical>
4.      <b-nav-item v-
    if="$store.state.jsonSessionStarted && !$store.state.userAuthenticated" to="/authentification" exact exact-
    active-class="active">
5.        Authentification
6.      </b-nav-item>
7.      <b-nav-item
8.        v-if="$store.state.jsonSessionStarted && $store.state.userAuthenticated && !$store.state.adminData"
9.        to="/get-admindata"
10.       exact
11.       exact-active-class="active"
12.     >
13.       Requête AdminData
14.     </b-nav-item>
15.     <b-nav-item v-if="$store.state.jsonSessionStarted && $store.state.userAuthenticated" to="/fin-
    session" exact exact-active-class="active">
16.       Fin session impôt
17.     </b-nav-item>
18.   </b-nav>
19. </template>
```

- ligne 4 : l'option **[Authentification]** n'est offerte que si la session jSON a démarré mais que l'utilisateur n'est pas authentifié. Si la session jSON n'a pas démarré ou que l'utilisateur est déjà authentifié alors l'option n'est pas offerte ;
- lignes 7-11 : l'option **[Requête AdminData]** n'est offerte que si la session jSON a démarré, que l'utilisateur est authentifié et qu'on n'a pas encore récupéré la donnée **[AdminData]**. Si l'une de ces trois conditions n'est pas satisfaite (session jSON pas démarrée, utilisateur pas authentifié ou la donnée **[AdminData]** déjà récupérée, l'option n'est pas offerte ;
- ligne 15 : l'option **[Fin session impôt]** est offerte dès que la session jSON a démarré et que l'utilisateur est authentifié, sinon elle ne l'est pas ;

4.16.3 Routage du serveur [nuxt]

Le routage du serveur est en général plus complexe que celle du client car l'utilisateur peut taper n'importe quelle URL dans la barre d'adresses de son navigateur. On peut laisser faire (après tout l'utilisateur n'est pas censé faire ça) ou essayer de contrôler les choses. C'est ce que nous allons faire ici, pour l'exemple, car dans le cas de l'application **[nuxt-12]**, on peut très bien s'en passer puisque le serveur de calcul de l'impôt est bien protégé contre ces URL à la main et sait envoyer les messages d'erreur adéquats. Nous l'avons vu dans **[next-12]** où il n'y avait aucun contrôle de routage.

Le routage d'un serveur **[nuxt]** est très différent d'un client **[nuxt]** quant à la notion de redirection :

- lorsque un serveur **[nuxt]** est redirigé, il envoie un ordre de redirection au navigateur client avec la cible de la redirection. Le navigateur fait alors une nouvelle requête au serveur **[nuxt]** en lui demandant la cible qui lui a été transmise. Tout se passe comme si l'utilisateur avait tapé à la main l'URL de la cible de la redirection : toute l'application **[nuxt]** redémarre et donc tout son cycle de vie (plugins serveur, store, routage serveur, pages) ;
- lorsqu'un client **[nuxt]** est redirigé, rien de tel n'arrive. Il y a un simple changement de page, le même que celui qui aurait été obtenu si l'utilisateur avait cliqué sur un lien menant à la cible de la redirection. Le cycle de vie est alors différent (routage client, affichage cible de la route) ;

Pour cette raison, il est préférable de séparer le routage client du routage serveur même si les deux codes peuvent paraître analogues.

Le script de routage du serveur **[middleware/server/routing]** sera le suivant :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
```

```
3.    // qui exécute ce code ?
4.    console.log('[middleware server], process.server', process.server, ', process.client=', process.client)
5.
6.    // on récupère quelques informations dans le store [nuxt]
7.    const store = context.store
8.    // d'où vient-on ?
9.    const from = store.state.from || 'nowhere'
10.   // où va-t-on ?
```

- dans le routage du client, la fonction de routage reçoit le contexte **[context]** avec la propriété **[context.from]** qui est la route de la page d'où l'on vient. La route où l'on va est obtenue par **[context.route]** ;
- dans le routage du serveur, la fonction de routage reçoit le contexte **[context] sans** la propriété **[context.from]**. Le routage du serveur n'intervient que lorsqu'une URL est demandée à la main au serveur **[nuxt]**. On sait qu'alors toute l'application **[nuxt]** est réinitialisée. C'est comme si on repartait de zéro et il n'y a donc pas de notion de 'page précédente' ;
- grâce à la session **[nuxt]** on sait que le serveur peut récupérer cette session et donc ne pas repartir de zéro. C'est donc dans cette session **[nuxt]** et plus particulièrement dans le store de cette session que nous stockerons le nom de la dernière page affichée par le navigateur client avant qu'une URL soit demandée au serveur **[nuxt]** ;
- lignes 7-9 : on récupère le nom de la dernière page affichée par le navigateur client. Au démarrage de l'application, cette information **[from]** n'existe pas dans le store. On affecte alors le nom **[nowhere]** à la variable **[from]** ;

Pour que le serveur **[nuxt]** puisse récupérer dans le store le nom de la dernière page affichée par le navigateur client, il faut que le client **[nuxt]** mette également cette information dans le store. Le script de routage du client **[nuxt]** est donc complété de la façon suivante :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware client], process.server', process.server, ', process.client=', process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session nuxt
7.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18.
19.   // on met dans la session le nom de la page où on va - pas de redirection serveur
20.   context.store.commit('replace', { serverRedirection: false, from: context.route.name })
21.   // on sauvegarde le store dans la session [nuxt]
22.   const session = context.app.$session()
23.   session.value.store = context.store.state
24.   session.save(context)
25. }
```

- les lignes 19-24 sont ajoutées ;
- ligne 20 : on met dans le store le nom de la page **[context.route.name]** qui va s'afficher et qui sera donc lors du routage suivant, la page d'où l'on vient. Par ailleurs, on va voir que dans le routage du serveur **[nuxt]**, celui-ci a besoin de savoir si le routage en cours est issu d'une précédente redirection du serveur **[nuxt]**. Ici ce n'est pas le cas, et on met donc la propriété **[serverRedirection]** à **[false]** ;
- lignes 22-24 : l'état du store est mis dans la session **[nuxt]** (ligne 23) puis la session **[nuxt]** est sauvegardée dans un cookie (ligne 24) qui lui-même sera sauvegardé dans le navigateur du client **[nuxt]** ;

Revenons sur le script de routage du serveur **[nuxt]** :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware server], process.server', process.server, ', process.client=', process.client)
5.
6.    // on récupère quelques informations dans le store [nuxt]
7.    const store = context.store
8.    // d'où vient-on ?
9.    const from = store.state.from || 'nowhere'
10.   // où va-t-on ?
11.   const to = context.route.name
12.   // éventuelle redirection
```

```
13.   let redirection = ''
14.   // gestion du routage terminé
15.   let done = false
16.
17.   // est-on déjà dans une redirection du serveur [nuxt]?
18.   if (store.state.serverRedirection) {
19.     // rien à faire
20.     done = true
21.   }
22.
23.   // est-ce un rechargement de page ?
24.   if (to === from) {
25.     // rien à faire
26.     done = true
27.   }
28.
29.   // contrôle de la navigation du serveur [nuxt]
30.   // on s'inspire de la navigation client dans le composant [navigation]
31.
32.   // cas où la session PHP n'a pas démarré
33.   if (!done && !store.state.jsonSessionStarted && to !== 'index') {
34.     // redirection
35.     redirection = 'index'
36.     // travail terminé
37.     done = true
38.   }
39.
40.   // cas où l'utilisateur n'est pas authentifié
41.   if (!done && store.state.jsonSessionStarted && !store.state.userAuthenticated && to !== 'authentification
    ') {
42.     // redirection
43.     redirection = from
44.     // travail terminé
45.     done = true
46.   }
47.
48.   // cas où l'utilisateur a été authentifié
49.   if (!done && store.state.jsonSessionStarted && store.state.userAuthenticated && to !== 'get-
    admindata' && to !== 'fin-session') {
50.     // on reste sur la même page
51.     redirection = from
52.     // travail terminé
53.     done = true
54.   }
55.
56.   // cas où [adminData] a été obtenu
57.   if (!done && store.state.jsonSessionStarted && store.state.userAuthenticated && store.state.adminData &&
    to !== 'fin-session') {
58.     // on reste sur la même page
59.     redirection = from
60.     // travail terminé
61.     done = true
62.   }
63.
64.   // on a fait tous les contrôles ---------------------
65.   // redirection ?
66.   if (redirection) {
67.     // on note la redirection dans le store
68.     store.commit('replace', { serverRedirection: true })
69.   } else {
70.     // pas de redirection
71.     store.commit('replace', { serverRedirection: false, from: to })
72.   }
73.   // on sauvegarde le store dans la session [nuxt]
74.   const session = context.app.$session()
75.   session.value.store = store.state
76.   session.save(context)
77.   // on fait l'éventuelle redirection
78.   if (redirection) {
79.     context.redirect({ name: redirection })
80.   }
81. }
```

- lignes 6-9 : on récupère la valeur de **[from]** dans le store du serveur **[nuxt]** ;
- ligne 11 : on note la cible du routage courant ;
- ligne 13 : le routage peut amener à une redirection du navigateur client. **[redirection]** sera la cible de cette redirection ;

- ligne 15 : **[done]** à **[true]** indique que le routage est terminé ;
- lignes 17-21 : on regarde d'abord si le routage courant est issu d'une demande de redirection envoyée au navigateur client. Cette information est stockée dans la propriété **[serverRedirection]** du store. Si cette propriété est à vrai, alors c'est que le serveur **[nuxt]** a envoyé une redirection au navigateur client lors de la précédente requête au serveur **[nuxt]**. Dans ce cas, il n'y a pas de routage à faire. Lors de la précédente requête, le routeur du serveur **[nuxt]** a décidé que le navigateur client devait être redirigé. Cette décision n'a pas à être remise en cause par un nouveau routage ;
- lignes 23-27 : on regarde si le routage en cours est un rechargement de page. Si oui, on laisse faire ;
- à partir de la ligne 29, on reprend les règles appliquées dans le composant **[navigation]** du client **[nuxt]** (cf paragraphe précédent) ;
- lignes 32-38 : on traite le cas où la session jSON n'a pas démarré et que la cible du routage n'est pas la page **[index]**. Dans ce cas, on redirige le navigateur client vers la page **[index]** ;
- lignes 40-46 : on traite le cas où la session jSON a démarré, l'utilisateur n'est pas authentifié et la cible du routage courant n'est pas la page **[authentification]**. Dans ce cas, on refuse le routage et on reste là où on était ;
- lignes 48-54 : on traite le cas où la session jSON a démarré, l'utilisateur est authentifié et la cible du routage courant n'est ni la page **[get-admindata]**, ni la page **[fin-session]** qui sont alors les seules destinations possibles. Dans ce cas, on refuse le routage demandé et on revient là où on était précédemment ;
- lignes 56-62 : on traite le cas où **[adminData]** a été obtenu. Dans ce cas, il n'y a qu'une cible possible pour le routage : la page **[fin-session]**. Si ce n'était pas elle qui était demandée, on refuse le routage et on revient là où on était précédemment ;
- lignes 64-72 : s'il y a eu redirection, on le note dans le store du serveur **[nuxt]** : **[serverRedirection: true]**. On notera qu'on ne donne pas de valeur à la propriété **[from]** du store. La raison en est qu'il va y avoir redirection du navigateur client et on a vu que dans ce cas, il n'y avait pas de routage (lignes 17-20) et la propriété **[from]** du store n'est pas utilisée ;
- lignes 66-69 : s'il n'y a pas de redirection, alors on le note également dans le store du serveur **[nuxt]** : **[serverRedirection: false]**. Par ailleurs, le routage en cours va afficher la page **[to]** qui pour la requête suivante (client ou serveur **[nuxt]**) deviendra la page précédente. C'est pourquoi on écrit **[from: to]** ;
- lignes 73-76 : on sauvegarde le store dans la session **[nuxt]** elle-même sauvegardée dans un cookie ;
- lignes 77-80 : si **[redirection]** n'est pas vide, alors on demande au navigateur de se rediriger. Sinon (on ne le voit pas ici), le cycle de vie du serveur **[nuxt]** va se poursuivre : la page **[to]** va être traitée par le serveur **[nuxt]** et envoyée au navigateur du client **[nuxt]** avec le cookie de session **[nuxt]** ;

Le routage choisi ici pour le serveur **[nuxt]** est arbitraire. On aurait pu en choisir un autre ou comme il a été dit ne pas en faire du tout. Celui choisi ci-dessus a le mérite de toujours laisser l'application dans un état stable quelque soit l'URL demandée par l'utilisateur.

On peut améliorer un point lorsque la page chargée au final est la page d'origine. Il y a deux cas :

- l'utilisateur a provoqué un rechargement de la page (to===from) ;
- il y a redirection vers la page d'origine (redirection===from) ;

Dans les deux cas la page d'origine va être de nouveau exécutée avec son appel asynchrone au serveur de calcul de l'impôt. Prenons un exemple. Si une fois authentifié, l'utilisateur recharge la page (F5). Dans ce cas dans le routage ci-dessus, on a : **[to]**=**[from]**=**[authentification]**. Il n'y a pas redirection. La page **[to=authentification]** va être exécutée par le serveur **[nuxt]**. Si on ne fait rien, la fonction **[asyncData]** va s'exécuter de nouveau. C'est inutile puisque l'authentification a déjà été faite.

On peut améliorer les choses en modifiant légèrement la page **[authentification]** :

```
1.    // données asynchrones
2.    async asyncData(context) {
3.      // log
4.      console.log('[authentification asyncData started]')
5.      // on ne fait pas les choses deux fois si la page a déjà été demandée
6.      if (process.server && context.store.state.userAuthenticated) {
7.        console.log('[authentification asyncData canceled]')
8.        return { result: '[succès]' }
9.      }
10.
11.     if (process.client) {
12.       // début attente
13.       context.app.$eventBus().$emit('loading', true)
14.       // pas d'erreur
15.       context.app.$eventBus().$emit('errorLoading', false)
16.     }
17.     try {
18.       // on s'authentifie auprès du serveur
19.       …
```

- lignes 6-9 : si la page est exécutée par le serveur **[nuxt]** et qu'on découvre dans le store que l'authentification a déjà été faite, alors on retourne directement le résultat souhaité (ligne 8) ;

On fait la même chose pour toutes les pages :

Page `[index]` :

```
1.    // données asynchrones
2.    async asyncData(context) {
3.      // log
4.      console.log('[index asyncData started]')
5.      // on ne fait pas les choses deux fois si la page a déjà été demandée
6.      if (process.server && context.store.state.jsonSessionStarted) {
7.        console.log('[index asyncData canceled]')
8.        return { result: '[succès]' }
9.      }
10.     try {
11.       …
```

Page `[get-admindata]`

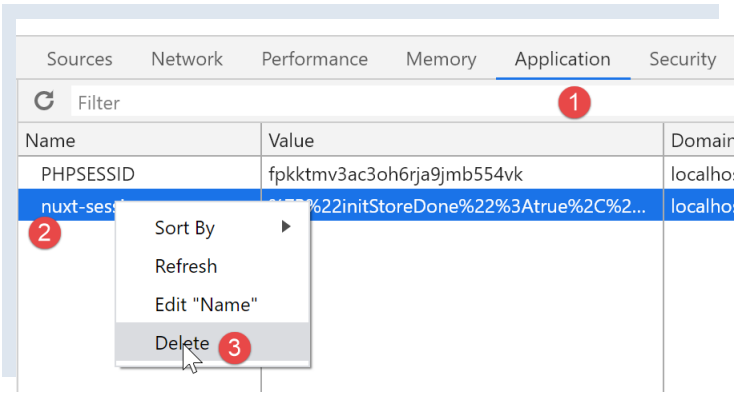
```
1.    // données asynchrones
2.    async asyncData(context) {
3.      // log
4.      console.log('[get-admindata asyncData started]')
5.      // on ne fait pas les choses deux fois si la page a déjà été demandée
6.      if (process.server && context.store.state.adminData) {
7.        console.log('[get-admindata asyncData canceled]')
8.        return { result: context.store.state.adminData }
9.      }
10.     // client
11.     if (process.client) {
12.       // début attente
13.       context.app.$eventBus().$emit('loading', true)
14.       // pas d'erreur
15.       context.app.$eventBus().$emit('errorLoading', false)
16.     }
17.     try {
18.       …
```

Page `[fin-session]`

```
1.    // données asynchrones
2.    async asyncData(context) {
3.      // log
4.      console.log('[fin-session asyncData started]')
5.      // on ne fait pas les choses deux fois si la page a déjà été demandée
6.      if (process.server && context.store.state.jsonSessionStarted && !context.store.state.userAuthenticated) {
7.        console.log('[fin-session asyncData canceled]')
8.        return { result: "[succès]. La session jSON reste initialisée mais vous n'êtes plus authentifié(e)." }
9.      }
10.     // cas du client [nuxt]
11.     if (process.client) {
12.       // début attente
13.       context.app.$eventBus().$emit('loading', true)
14.       // pas d'erreur
15.       context.app.$eventBus().$emit('errorLoading', false)
16.     }
17.     try {
18.
```

4.16.4 Exécution

Pour exécuter cet exemple, il faut prendre soin avant l'exécution de supprimer le cookie de session **`[nuxt]`** et le cookie PHP du navigateur exécutant le client **`[nuxt]`** afin de partir d'une situation nette. Ci-dessous un exemple avec le navigateur Chrome :



4.16.5 Conclusion

Le routage du serveur **[nuxt]** est complexe car il faut prévoir toutes les URL que peut taper à la main l'utilisateur. C'est un cas d'école. Une application **[nuxt]** n'est pas destinée à être utilisée de cette façon. Une fois la page **[index]** servie par le routeur du serveur **[nuxt]**, on pourrait rediriger les appels suivants faits au serveur vers une page d'erreur.

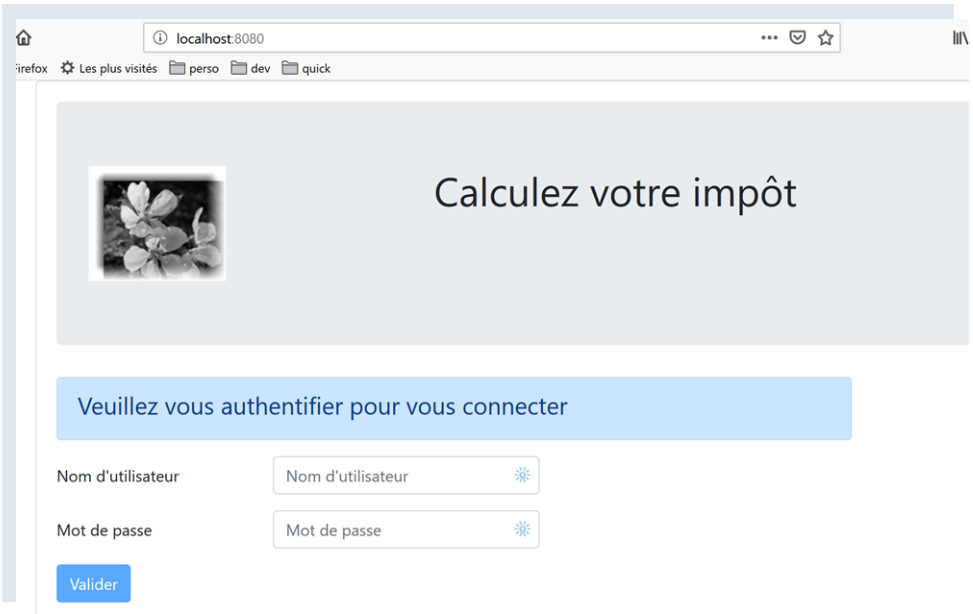
Dans le cas précis de notre exemple **[nuxt-13]**, le routage du serveur **[nuxt]** était inutile. Celui fait par défaut (absence de routage en fait) dans l'exemple **[nuxt-12]** convenait très bien.

4.17 Exemple [nuxt-20] : portage de l'exemple [vuejs-22]

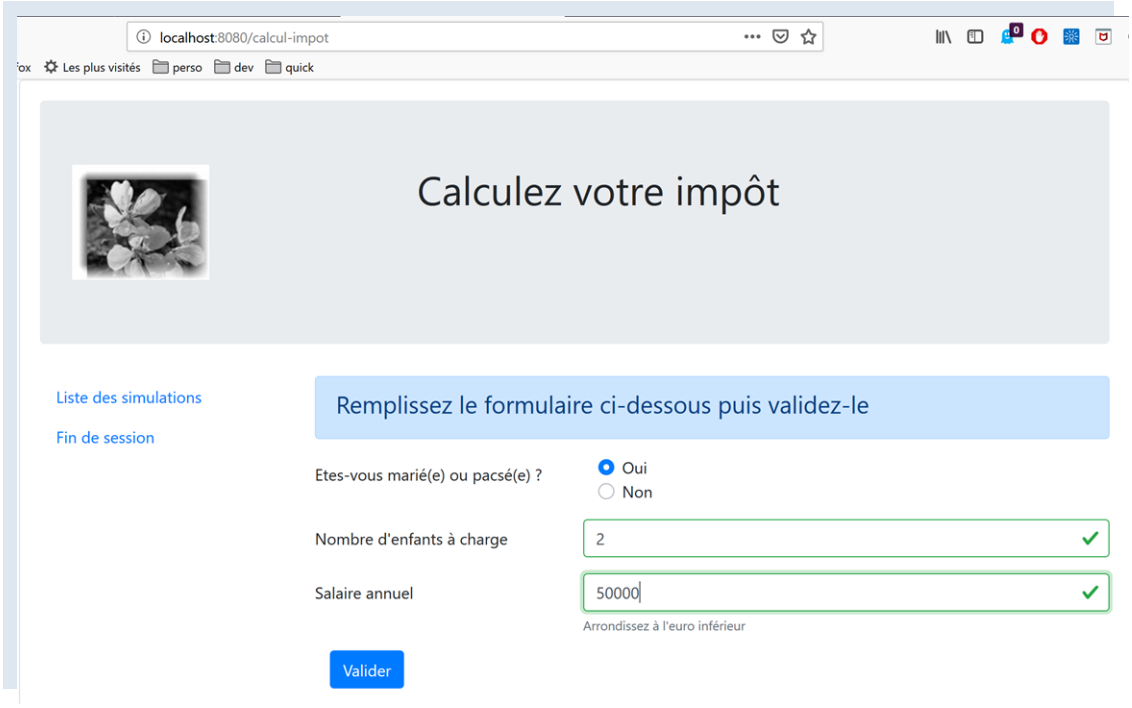
4.17.1 Présentation

Nous nous proposons ici de porter l'exemple **[vuejs-22]** qui était une application **[vue.js]** de type SPA, dans un contexte **[nuxt]** SSR. **[vuejs-22]** était une application cliente du serveur de calcul de l'impôt qui présentait les vues suivantes :

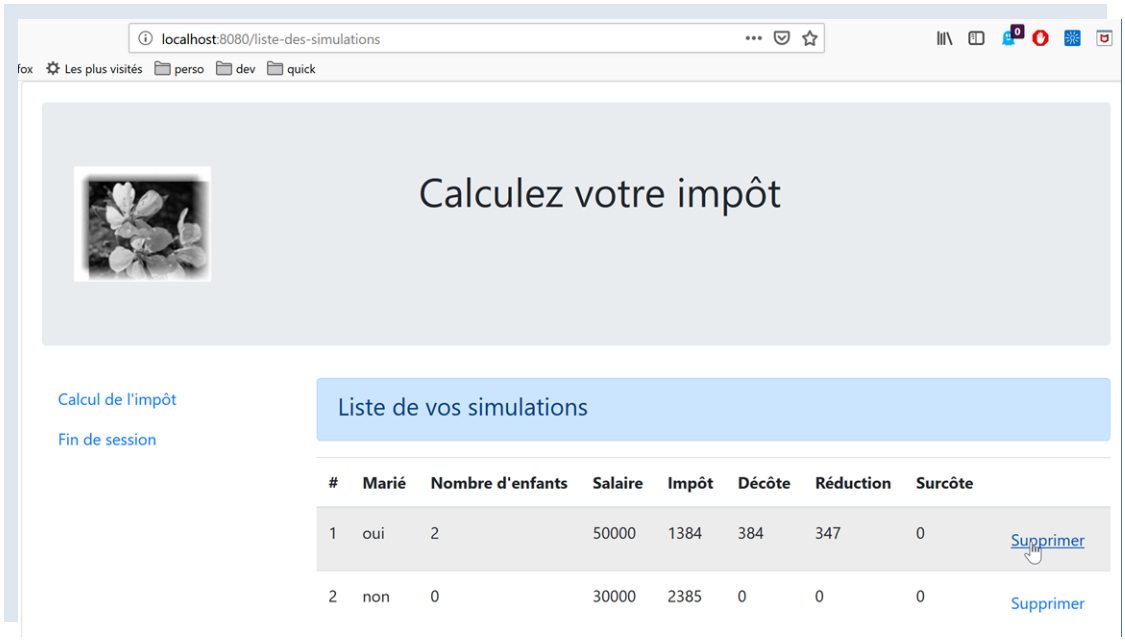
La 1ère vue est la vue d'authentification :



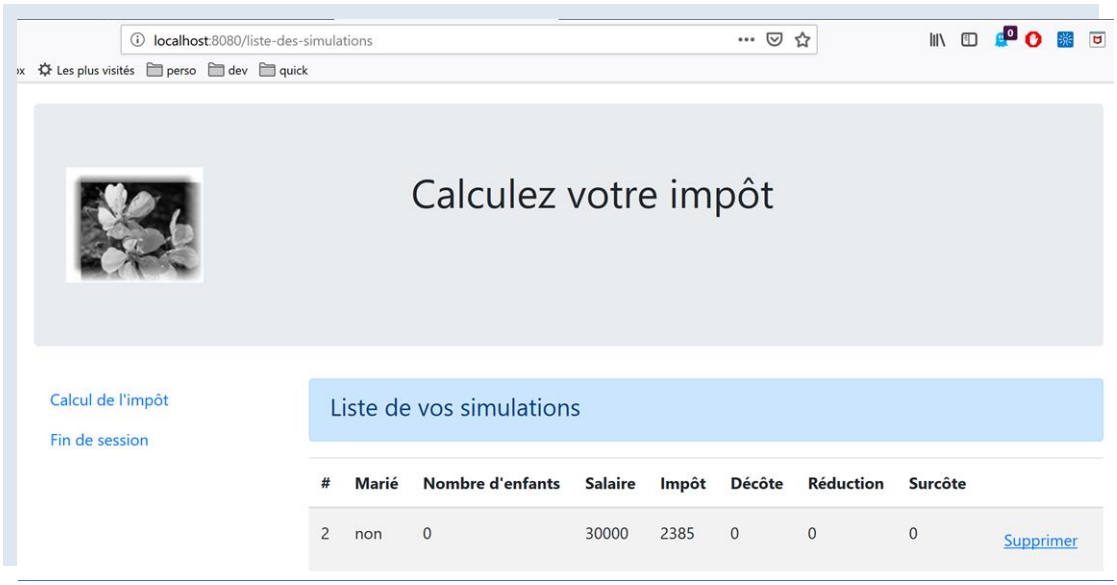
La seconde vue est celle du calcul de l'impôt :



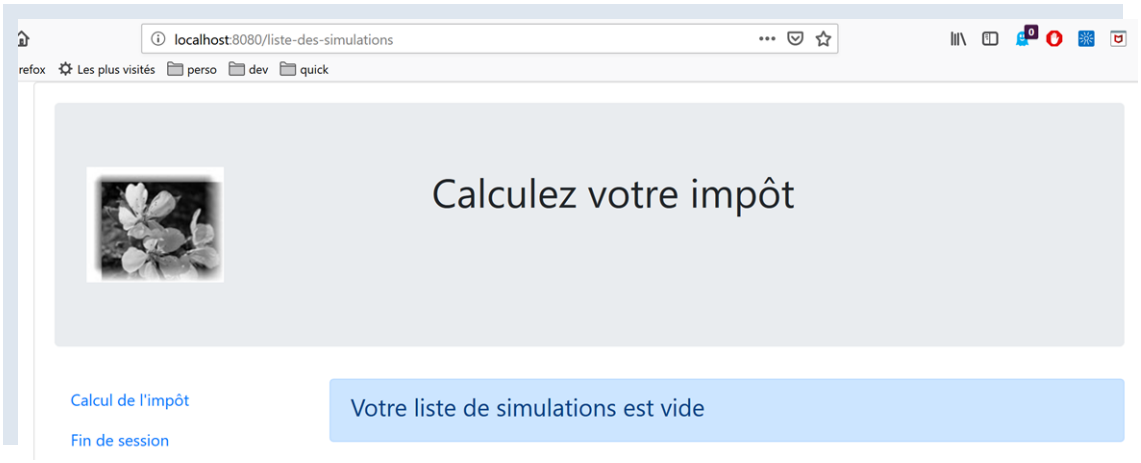
La 3ième vue est celle qui affiche la liste des simulations faites par l'utilisateur :



L'écran ci-dessus montre qu'on peut supprimer la simulation n° 1. On obtient alors la vue suivante :



Si on supprime maintenant la dernière simulation, on obtient la nouvelle vue suivante :



Nous allons porter l'application **[vuejs-22]** vers l'application **[nuxt-20]** de façon progressive. Nous n'expliquerons pas de nouveau les codes de **[vuejs-22]**. Le lecteur est invité à relire le document | Introduction au framework **VUE.JS** par l'exemple |. Les différentes étapes devraient montrer les différences entre une application **[vuejs]** et une application **[nuxt]**.

4.17.2 étape 1

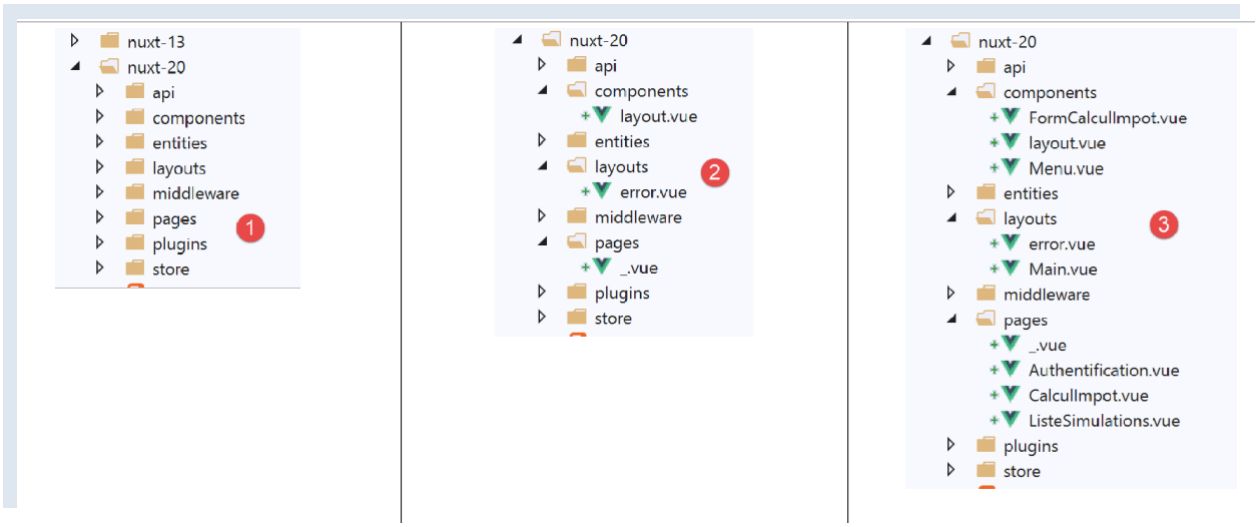
Le projet **[nuxt-20]** est initialement obtenu par recopie du projet **[nuxt-12]**. Celui-ci est en effet un bon point de départ :

- il sait dialoguer avec le serveur de calcul de l'impôt ;
- il gère correctement les erreurs que celui-ci envoie ;
- les client et serveur **[nuxt]** savent communiquer via une session **[nuxt]** ;

On a donc une bonne infrastructure de départ. Notre principal travail devrait être de modifier :

- les pages. On prendra celles du projet **[vuejs-22]** qu'il faudra adapter au nouvel environnement ;
- la gestion du store. Des informations supplémentaires devraient apparaître (liste des simulations) et d'autres pourraient devenir inutiles ;
- la gestion du routage du client et du serveur **[nuxt]** ;

Donc tout d'abord, on crée le projet **[nuxt-20]** en recopiant le projet **[nuxt-12]** :



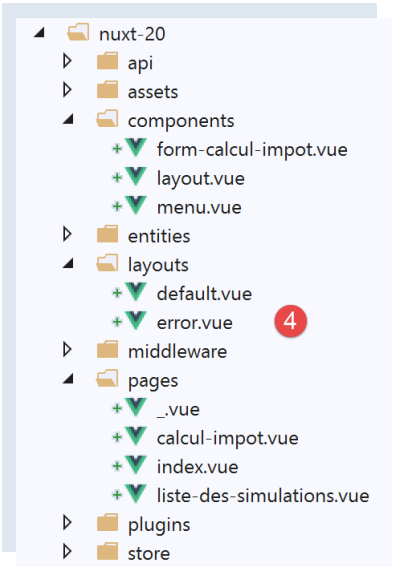
Puis on supprime les pages et composants devenus inutiles **[2]** :

- le composant **[components/navigation]** disparaît ;
- le layout **[layout/default]** disparaît ;
- les pages **[index, authentification, get-admindata, fin-session]** disparaissent ;

Puis on intègre dans **[nuxt-20]** des éléments de **[vuejs-22] [3]** :

- les trois pages **[Authentification, CalculImpot, ListeSimulations]** de l'application **[vuejs-22]** vont dans le dossier **[pages]** ;
- les composants **[FormCalculImpot, Menu, Layout]** de l'application **[vuejs-22]** vont dans le dossier **[components]** ;
- la page **[Main]** de **[vuejs-22]** qui servait de **[layout]** à l'application **[vuejs-22]** va dans le dossier **[layouts]** ;

On renomme les éléments intégrés **[4]** :



- dans **[layouts]**, **[Main]** est devenue **[default]** puisque c'est le nom par défaut du layout d'une application **[nuxt]** ;
- dans **[pages]**, la page **[Authentification]** est devenue **[index]**, car **[Authentification]** jouait ce rôle dans l'application **[vuejs-22]** ;

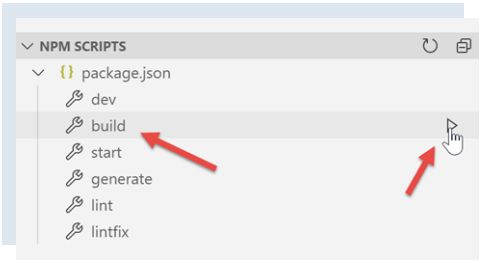
A ce point là, on peut faire une compilation du projet pour voir les premières erreurs. On modifie le fichier **[nuxt.config]** de l'exemple **[nuxt-12]** afin d'exécuter désormais **[nuxt-20]** :

```
1.  export default {
2.    mode: 'universal',
3.    /*
```

```
4.      ** Headers of the page
5.      */
6.     head: {
7.       title: 'Introduction à [nuxt.js]',
8.       meta: [
9.         { charset: 'utf-8' },
10.        { name: 'viewport', content: 'width=device-width, initial-scale=1' },
11.        {
12.          hid: 'description',
13.          name: 'description',
14.          content: 'ssr routing loading asyncdata middleware plugins store'
15.        }
16.      ],
17.      link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
18.    },
19.    /*
20.     ** Customize the progress-bar color
21.     */
22.    loading: false,
23.
24.    /*
25.     ** Global CSS
26.     */
27.    css: [],
28.    /*
29.     ** Plugins to load before mounting the App
30.     */
31.    plugins: [
32.      { src: '@/plugins/client/plgSession', mode: 'client' },
33.      { src: '@/plugins/server/plgSession', mode: 'server' },
34.      { src: '@/plugins/client/plgDao', mode: 'client' },
35.      { src: '@/plugins/server/plgDao', mode: 'server' },
36.      { src: '@/plugins/client/plgEventBus', mode: 'client' },
37.      { src: '@/plugins/client/plgMétier', mode: 'client' }
38.    ],
39.    /*
40.     ** Nuxt.js dev-modules
41.     */
42.    buildModules: [
43.      // Doc: https://github.com/nuxt-community/eslint-module
44.      '@nuxtjs/eslint-module'
45.    ],
46.    /*
47.     ** Nuxt.js modules
48.     */
49.    modules: [
50.      // Doc: https://bootstrap-vue.js.org
51.      'bootstrap-vue/nuxt',
52.      // Doc: https://axios.nuxtjs.org/usage
53.      '@nuxtjs/axios',
54.      // https://www.npmjs.com/package/cookie-universal-nuxt
55.      'cookie-universal-nuxt'
56.    ],
57.    /*
58.     ** Axios module configuration
59.     ** See https://axios.nuxtjs.org/options
60.     */
61.    axios: {},
62.    /*
63.     ** Build configuration
64.     */
65.    build: {
66.      /*
67.       ** You can extend webpack config here
68.       */
69.      extend(config, ctx) {}
70.    },
71.    // répertoire du code source
72.    srcDir: 'nuxt-20',
73.    // routeur
74.    router: {
75.      // racine des URL de l'application
76.      base: '/nuxt-20/',
77.      // middleware de routage
78.      middleware: ['routing']
79.    },
80.    // serveur
```

```
81.   server: {
82.     // port de service, 3000 par défaut
83.     port: 81,
84.     // adresses réseau écoutées, par défaut localhost : 127.0.0.1
85.     // 0.0.0.0 = toutes les adresses réseau de la machine
86.     host: 'localhost'
87.   },
88.   // environnement
89.   env: {
90.     // configuration axios
91.     timeout: 2000,
92.     withCredentials: true,
93.     baseURL: 'https://localhost/php7/scripts-web/impots/version-14',
94.     // configuration du cookie de session [nuxt]
95.     maxAge: 60 * 5
96.   }
97. }
```

On fait ensuite un **[build]** du projet :

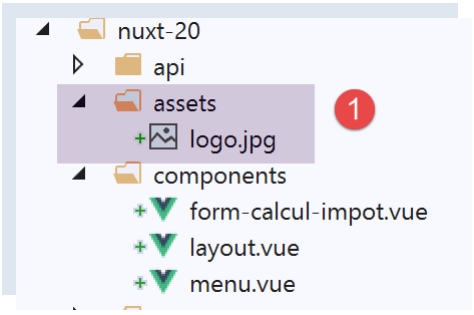


Les erreurs signalées sont les suivantes :

```
1.  Module not found: Error: Can't resolve '../assets/logo.jpg' @ ./nuxt-20/layouts/default.vue?...
2.  Module not found: Error: Can't resolve './FormCalculImpot' @ ./nuxt-20/pages/calcul-impot.vue?...
3.  Module not found: Error: Can't resolve './Layout' @ ./nuxt-20/pages/_.vue?...
4.  Module not found: Error: Can't resolve './Layout' @ ./nuxt-20/pages/liste-des-simulations...
5.  Module not found: Error: Can't resolve './Layout' @ ./nuxt-20/pages/calcul-impot.vue?...
6.  Module not found: Error: Can't resolve './Menu' @ ./nuxt-20/pages/_.vue?...
7.  Module not found: Error: Can't resolve './Menu' @ ./nuxt-20/pages/calcul-impot.vue
```

- l'erreur de la ligne 1 indique qu'on référence une image inexistante. On la récupèrera dans **[vuejs-22]** ;
- l'erreur de la ligne 2 montre que le composant **[./FormCalculImpot]** n'existe pas. Effectivement, ce composant est désormais dans **[@/components/form-calcul-impot]** ;
- les erreurs des lignes **[3-5]** montrent que le composant **[./Layout]** n'existe pas. Effectivement, ce composant est désormais dans **[@/components/layout]** ;
- les erreurs des lignes **[6-7]** montrent que le composant **[./Menu]** n'existe pas. Effectivement, il s'appelle maintenant **[@/components/menu]** ;

On ajoute l'image **[assets/logo.jpg]** au projet **[nuxt-20]** :



Par ailleurs, dans toutes les pages on va corriger le chemin des composants. Prenons l'exemple de la page **[calcul-impot]** :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <Layout :left="true" :right="true">
5.        <!-- formulaire de calcul de l'impôt à droite -->
6.        <FormCalculImpot slot="right" @resultatObtenu="handleResultatObtenu" />
7.        <!-- menu de navigation à gauche -->
```

```
8.        <Menu slot="left" :options="options" />
9.      </Layout>
10.     <!-- zone d'affichage des résultat du calcul de l'impôt sous le formulaire -->
11.     <b-row v-if="résultatObtenu" class="mt-3">
12.       <!-- zone de trois colonnes vide -->
13.       <b-col sm="3" />
14.       <!-- zone de neuf colonnes -->
15.       <b-col sm="9">
16.         <b-alert show variant="success">
17.           <span v-html="résultat"></span>
18.         </b-alert>
19.       </b-col>
20.     </b-row>
21.   </div>
22. </template>
23.
24. <script>
25. // imports
26. import FormCalculImpot from './FormCalculImpot'
27. import Menu from './Menu'
28. import Layout from './Layout'
29.
30. export default {
31.   // composants utilisés
32.   components: {
33.     Layout,
34.     FormCalculImpot,
35.     Menu
36.   },
37.
```

Les trois **[import]** des lignes 26-28 deviennent :

```
1.  // imports
2.  import FormCalculImpot from '@/components/form-calcul-impot'
3.  import Menu from '@/components/menu'
4.  import Layout from '@/components/layout'
```

On vérifie et éventuellement corrige ainsi les **[import]** de tous les composants, layouts et pages. Une fois ces corrections faites, on peut tenter un nouveau **[build]**. Normalement il n'y a plus d'erreurs.

On peut alors tenter une exécution :

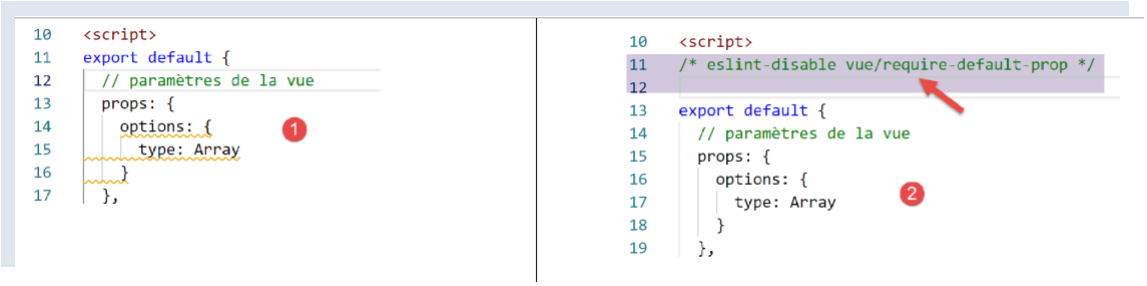
Des erreurs apparaissent :

```
1.  Module Error (from ./node_modules/eslint-loader/dist/cjs.js):
2.
3.  c:\Data\st-2019\dev\nuxtjs\dvp\nuxt-20\pages\index.vue
4.     92:29  error  Expected '!==' and instead saw '!='  eqeqeq
5.    129:27  error  Expected '!==' and instead saw '!='  eqeqeq
6.  150:28  error  Expected '!==' and instead saw '!='  eqeqeq
```

On voit que la commande **[dev]** alliée au module **[eslint]** est plus stricte, au niveau syntaxique, que la commande **[build]**. Ici, elle réclame que l'opérateur de comparaison **[!=]** soit écrit **[!==]** qui est un opérateur plus strict (il vérifie également le type des opérandes). Ces erreurs se produisent dans la page **[index.vue]**.

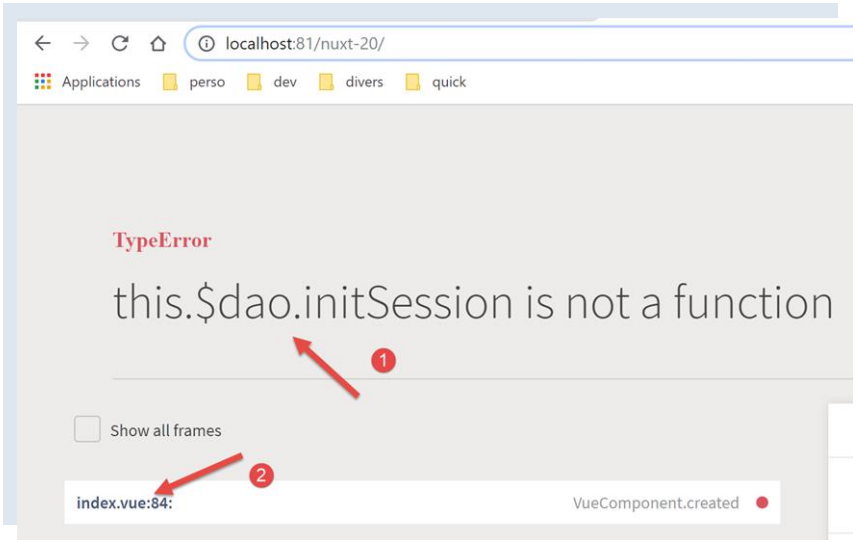
On corrige les erreurs ci-dessus et on relance l'exécution du projet. On a alors un warning du module **[eslint]** :

```
1.  c:\Data\st-2019\dev\nuxtjs\dvp\nuxt-20\components\menu.vue
2.  14:5  warning  Prop 'options' requires default value to be set  vue/require-default-prop
```



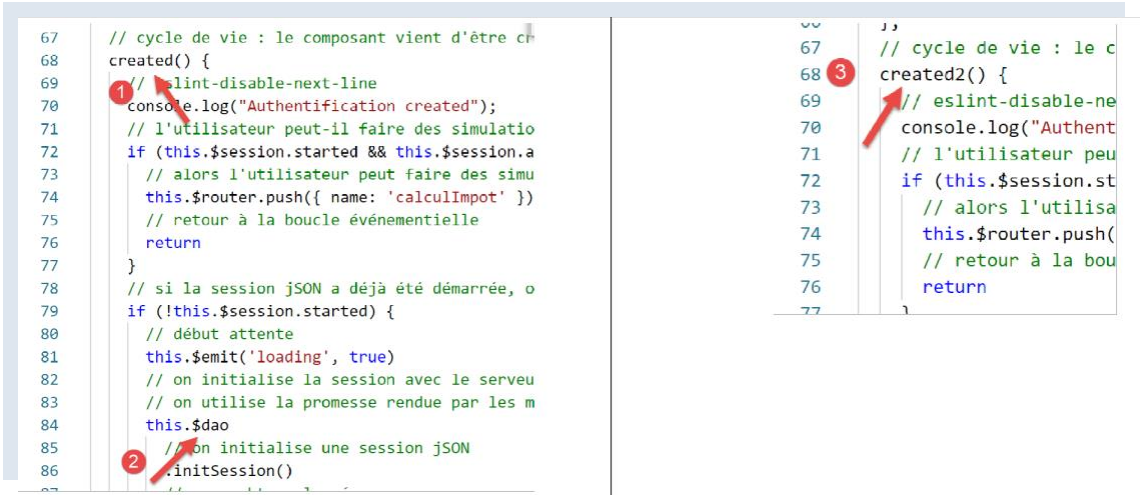
On corrige cette erreur avec le **[Quick fix]** du module **[eslint] [2]**.

On relance l'exécution du projet. On n'a plus d'erreur de compilation. On demande alors l'URL **[http://localhost:81/nuxt-20/]** avec un navigateur. On obtient une erreur d'exécution :



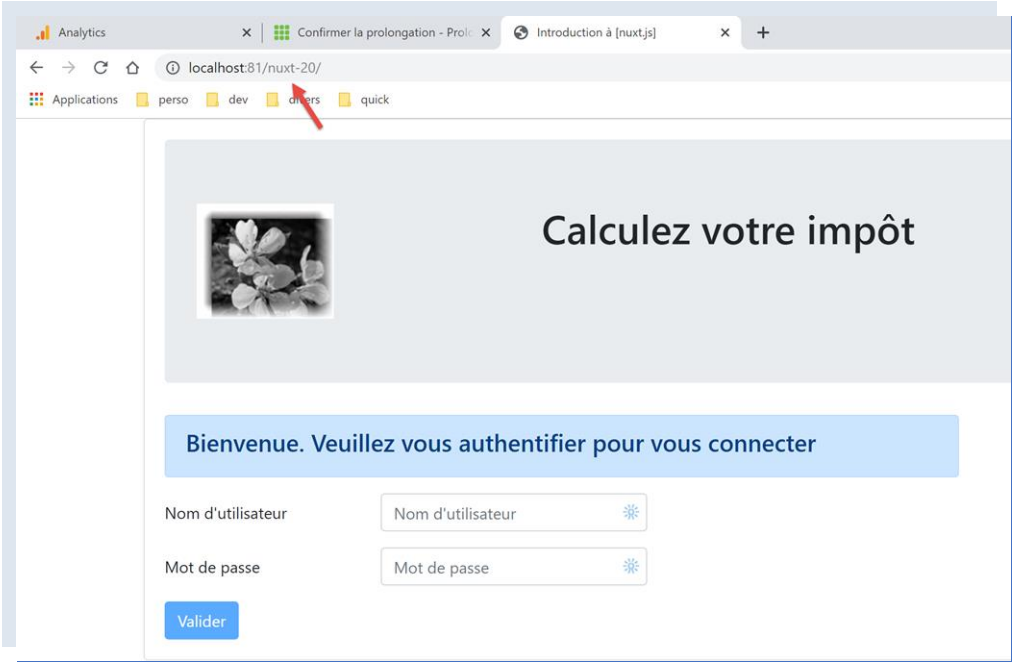
L'erreur se trouve dans **[index.vue] [2]**. L'erreur **[1]** vient du fait que dans **[vuejs-22]**, la couche **[dao]** était disponible dans **[this.$dao]** alors que dans **[nuxt-12]** dont nous avons adopté l'infrastructure, elle est disponible dans la fonction **[this.$dao()]**.

L'erreur est dans la fonction **[created]** du cycle de vie de la page **[index]** :



Pour l'instant, on se contente de renommer **[created]** en **[created2]** pour que la fonction du cycle de vie **[created]** ne soit pas exécutée **[3]**.

On sauve la modification et on recharge la page **[index]** dans le navigateur. Cette fois-ci c'est bon :



4.17.3 étape 2

Les pages du projet **[vuejs-22]** utilisaient les éléments injectés suivants :

- $**dao** : pour la couche **[dao]** du client **[vue.js]** ;
- $**session** : pour une session stockée dans le **[localStorage]** du navigateur ;

Ces éléments n'existent plus dans l'infrastructure du projet **[nuxt-12]** que nous avons recopiée :

- il y a désormais deux couches **[dao]**, l'une pour le client **[nuxt]**, l'autre pour le serveur **[nuxt]**. Toutes deux sont disponibles via une **fonction** injectée appelée **[$dao]**. Cela signifie que dans les pages de l'application **[this.$dao]** doit être remplacé par **[this.$dao()]** ;
- la session **[nuxt]** gérée par l'application **[nuxt-20]** n'a plus rien à voir avec l'objet **[$session]** de l'application **[vuejs-22]** où il n'y avait pas de notion de cookie de session. Néanmoins, elles ont une fonctionnalité analogue : stocker des informations persistantes au fil des actions de l'utilisateur. La session **[nuxt]** stocke les informations dans le store plutôt que directement dans la session. Dans les pages de l'application **[this.$session]** doit être remplacé par **[this.$store]** lorsqu'il s'agit de mémoriser des informations dans la session et par **[this.$session()]** lorsqu'il s'agit de manipuler la session elle-même ;
- pour connaître l'état d'une propriété P du store, il faudra écrire **[this.$store.state.P]** ;
- pour changer la propriété P du store, il faudra écrire **[this.$store.commit('replace', {P:value})]** ;

Nous faisons ces modifications dans la page **[index]** :

```
<!-- définition HTML de la vue -->
<template>
  <Layout :left="false" :right="true">
    <template slot="right">
      <!-- formulaire HTML - on poste ses valeurs avec l'action [authentifier-utilisateur] -->
      <b-form @submit.prevent="login">
        <!-- titre -->
        <b-alert show variant="primary">
          <h4>Bienvenue. Veuillez vous authentifier pour vous connecter</h4>
        </b-alert>
        <!-- 1ère ligne -->
        <b-form-group label="Nom d'utilisateur" label-for="user" label-cols="3">
          <!-- zone de saisie user -->
          <b-col cols="6">
            <b-form-input id="user" v-model="user" type="text" placeholder="Nom d'utilisateur" />
          </b-col>
        </b-form-group>
        <!-- 2ième ligne -->
        <b-form-group label="Mot de passe" label-for="password" label-cols="3">
          <!-- zone de saisie password -->
          <b-col cols="6">
            <b-input id="password" v-model="password" type="password" placeholder="Mot de passe" />
```

```
          </b-col>
        </b-form-group>
        <!-- 3ième ligne -->
        <b-alert v-if="showError" show variant="danger" class="mt-3">L'erreur suivante s'est produite : {{ message }}</b-alert>
        <!-- bouton de type [submit] sur une 3ième ligne -->
        <b-row>
          <b-col cols="2">
            <b-button :disabled="!valid" variant="primary" type="submit">Valider</b-button>
          </b-col>
        </b-row>
      </b-form>
    </template>
  </Layout>
</template>

<!-- dynamique de la vue -->
<script>
/* eslint-disable no-console */
import Layout from '@/components/layout'
export default {
  // composants utilisés
  components: {
    Layout
  },
  // état du composant
  data() {
    return {
      // utilisateur
      user: '',
      // son mot de passe
      password: '',
      // contrôle l'affichage d'un msg d'erreur
      showError: false,
      // le message d'erreur
      message: ''
    }
  },

  // propriétés calculées
  computed: {
    // saisies valides
    valid() {
      return this.user && this.password && this.$store.state.started
    }
  },
  // cycle de vie : le composant vient d'être créé
  mounted() {
    // eslint-disable-next-line
    console.log("Authentification mounted");
    // l'utilisateur peut-il faire des simulations ?
    if (this.$store.state.started && this.$store.state.authenticated && this.$métier.taxAdminData) {
      // alors l'utilisateur peut faire des simulations
      this.$router.push({ name: 'calculImpot' })
      // retour à la boucle événementielle
      return
    }
    // si la session jSON a déjà été démarrée, on ne la redémarre pas de nouveau
    if (!this.$store.state.started) {
      // début attente
      this.$emit('loading', true)
      // on initialise la session avec le serveur - requête asynchrone
      // on utilise la promesse rendue par les méthodes de la couche [dao]
      this.$dao()
        // on initialise une session jSON
        .initSession()
        // on a obtenu la réponse
        .then((response) => {
          // fin attente
          this.$emit('loading', false)
          // analyse de la réponse
          if (response.état !== 700) {
            // on affiche l'erreur
            this.message = response.réponse
            this.showError = true
            // retour à la boucle événementielle
            return
```

```
          }
          // la session a démarré
          this.$store.commit('replace', { started: true })
          console.log('[authentification], session=', this.$session())
        })
        // en cas d'erreur
        .catch((error) => {
          // on remonte l'erreur à la vue [Main]
          this.$emit('error', error)
        })
        // dans tous les cas
        .finally(() => {
          // on sauvegarde la session
          this.$session().save()
        })
    }
  },

  // gestionnaires d'évts
  methods: {
    // ----------- authentification
    async login() {
      try {
        // début attente
        this.$emit('loading', true)
        // on n'est pas encore authentifié
        this.$store.commit('replace', { authenticated: false })
        // authentification bloquante auprès du serveur
        const response = await this.$dao().authentifierUtilisateur(this.user, this.password)
        // fin du chargement
        this.$emit('loading', false)
        // analyse de la réponse du serveur
        if (response.état !== 200) {
          // on affiche l'erreur
          this.message = response.réponse
          this.showError = true
          // retour à la boucle événementielle
          return
        }
        // pas d'erreur
        this.showError = false
        // on est authentifié
        this.$store.commit('replace', { authenticated: true })
        // --------- on demande maintenant les données de l'administration fiscale
        // au départ, pas de donnée
        this.$métier.setTaxAdminData(null)
        // début attente
        this.$emit('loading', true)
        // demande bloquante auprès du serveur
        const response2 = await this.$dao().getAdminData()
        // fin du chargement
        this.$emit('loading', false)
        // analyse de la réponse
        if (response2.état !== 1000) {
          // on affiche l'erreur
          this.message = response2.réponse
          this.showError = true
          // retour à la boucle événementielle
          return
        }
        // pas d'erreur
        this.showError = false
        // on mémorise dans la couche [métier] la donnée reçue
        this.$métier.setTaxAdminData(response2.réponse)
        // on peut passer au calcul de l'impôt
        this.$router.push({ name: 'calculImpot' })
      } catch (error) {
        // on remonte l'erreur au composant principal
        this.$emit('error', error)
      } finally {
        // maj store
        this.$store.commit('replace', { métier: this.$métier })
        // on sauvegarde la session
        this.$session().save()
      }
    }
  }
```

```
176.}
177.</script>
```

Notons les points suivants :

- ligne 69 : la fonction **[created2]** a été renommée **[mounted]**, ceci pour que le serveur **[nuxt]** ne l'exécute pas (il n'exécute ni **[beforeMount]** ni **[mounted]**). Seul le client **[nuxt]** l'exécutera comme c'était le cas avec l'exemple **[vuejs-22]** ;
- ligne 73 : on référence **[this.$métier]** qui pour l'instant n'existe pas ;
- ligne 75 : nous n'avons jamais utilisé cette méthode dans une application **[nuxt]**. Il faudra voir si elle fonctionne dans un contexte **[nuxt]** ;
- ligne 112, 172 : dans **[vuejs-22]**, la session du projet était sauvegardée de cette façon. Avec le projet **[nuxt-20]**, la méthode **[save]** doit recevoir le contexte courant. On sait que dans une page **[nuxt]**, l'objet **[context]** est disponible dans **[this.$nuxt.context]** ;

Les lignes 112 et 172 sont donc réécrites de la façon suivante :

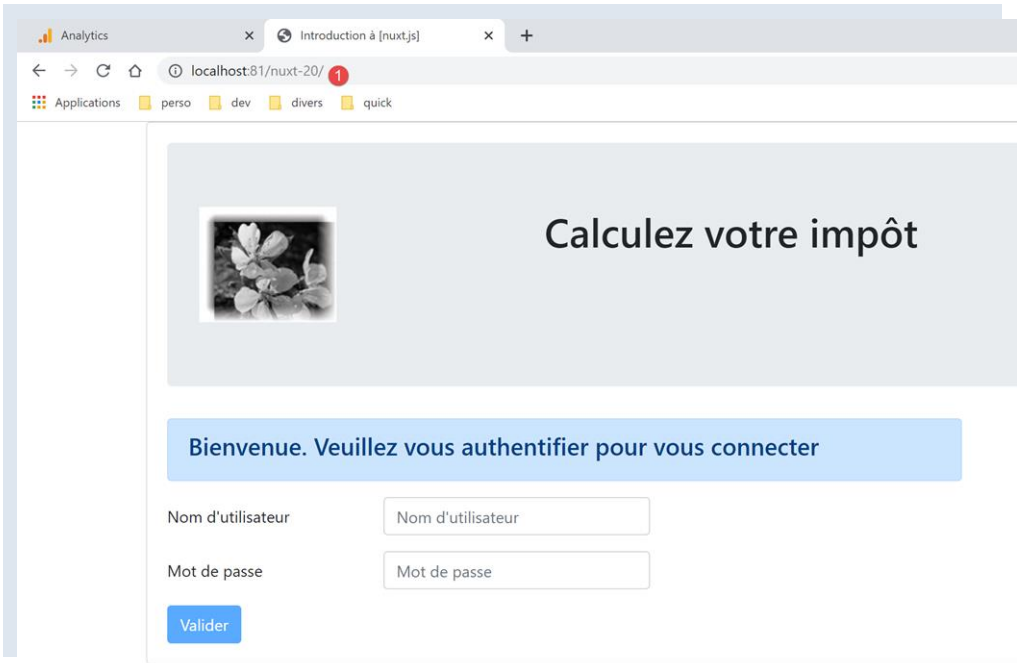
```
this.$session().save(this.$nuxt.context)
```

On notera que ce code n'est pas optimisé. Plutôt que d'utiliser plusieurs fois la fonction **[this.$session()]**, il serait préférable d'écrire :

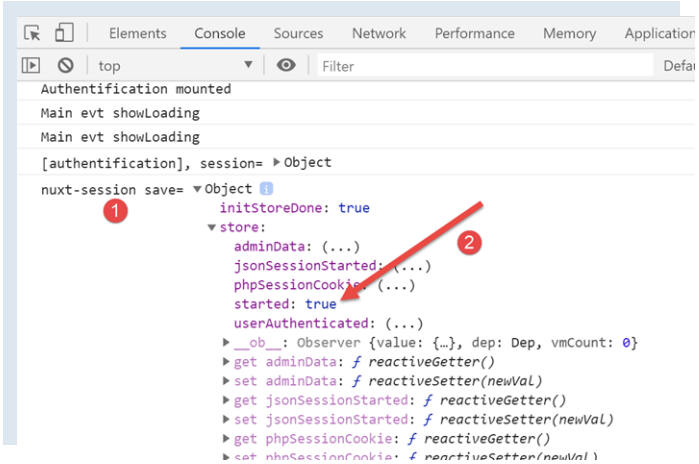
```
const session=this.$session()
```

puis utiliser ensuite la variable **[session]**. On peut tenir le même raisonnement pour la fonction **[this.$dao()]**.

Ces corrections faites, nous pouvons recharger l'URL **[http://localhost:81/nuxt-20/]** avec un navigateur. Nous obtenons toujours la même page que précédemment :

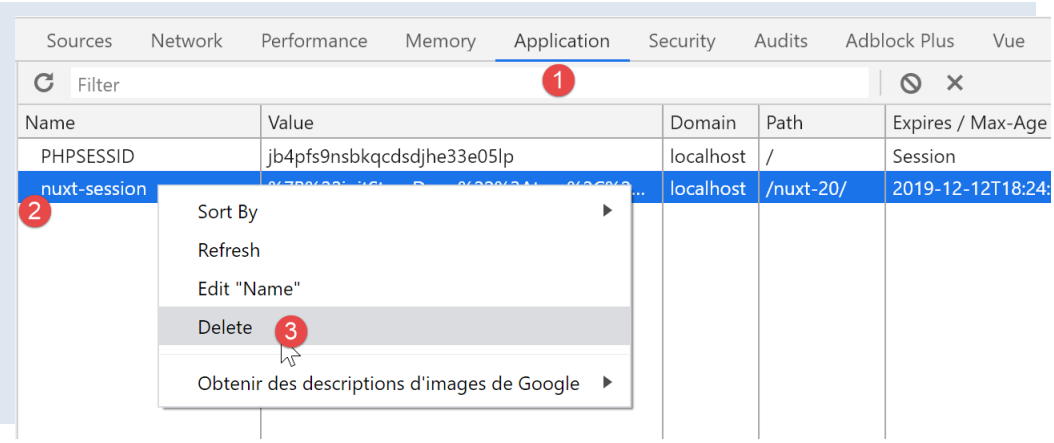


Regardons les logs du navigateur :

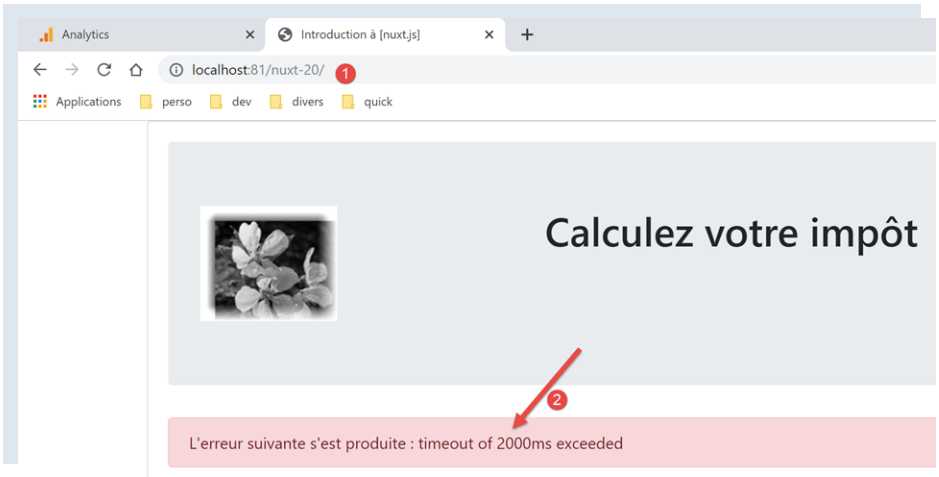


Le log **[1]** est le dernier log fait par le client **[nuxt]**. En **[2]**, on voit que la propriété **[started]** est à **[vrai]**, ce qui veut dire que la fonction **[mounted]** a réussi à démarrer une session jSON avec le serveur de calcul de l'impôt. On voit également que le store a des propriétés qu'il faudra soit abandonner soit renommer. Rappelons que nous utilisons le store de l'exemple **[nuxt-12]**.

Maintenant redemandons l'URL **[http://localhost:81/nuxt-20/]** alors que le serveur de calcul de l'impôt n'est pas lancé. On prend soin tout d'abord de supprimer le cookie de la session **[nuxt]** :



La copie d'écran ci-dessus est une copie d'écran Chrome. Ceci fait, l'URL **[http://localhost:81/nuxt-20/]** donne le résultat suivant :



L'erreur était correctement gérée par le projet **[vuejs-22]**. Elle reste correctement gérée par le projet **[nuxt-20]**.

4.17.4 étape 3

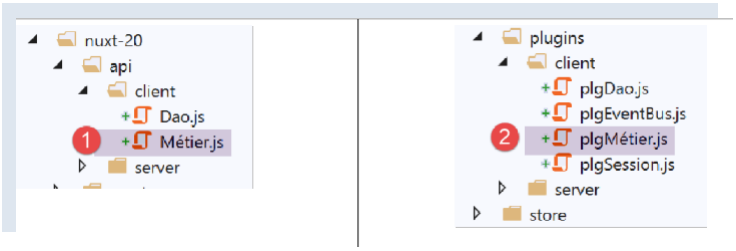
Maintenant que nous avons la page d'authentification, il faut regarder le code exécuté lorsque l'utilisateur clique sur le bouton **[Valider]** :

```
1.  // gestionnaires d'évts
2.    methods: {
3.      // ---------- authentification
4.      async login() {
5.        try {
6.          // début attente
7.          this.$emit('loading', true)
8.          // on n'est pas encore authentifié
9.          this.$store.commit('replace', { authenticated: false })
10.         // authentification bloquante auprès du serveur
11.         const response = await this.$dao().authentifierUtilisateur(this.user, this.password)
12.         // fin du chargement
13.         this.$emit('loading', false)
14.         // analyse de la réponse du serveur
15.         if (response.état !== 200) {
16.           // on affiche l'erreur
17.           this.message = response.réponse
18.           this.showError = true
19.           // retour à la boucle événementielle
20.           return
21.         }
22.         // pas d'erreur
23.         this.showError = false
24.         // on est authentifié
25.         this.$store.commit('replace', { authenticated: true })
26.         // --------- on demande maintenant les données de l'administration fiscale
27.         // au départ, pas de donnée
28.         this.$métier.setTaxAdminData(null)
29.         // début attente
30.         this.$emit('loading', true)
31.         // demande bloquante auprès du serveur
32.         const response2 = await this.$dao().getAdminData()
33.         // fin du chargement
34.         this.$emit('loading', false)
35.         // analyse de la réponse
36.         if (response2.état !== 1000) {
37.           // on affiche l'erreur
38.           this.message = response2.réponse
39.           this.showError = true
40.           // retour à la boucle événementielle
41.           return
42.         }
43.         // pas d'erreur
44.         this.showError = false
45.         // on mémorise dans la couche [métier] la donnée reçue
46.         this.$métier.setTaxAdminData(response2.réponse)
47.         // on peut passer au calcul de l'impôt
48.         this.$router.push({ name: 'calculImpot' })
49.       } catch (error) {
50.         // on remonte l'erreur au composant principal
51.         this.$emit('error', error)
52.       } finally {
53.         // maj store
54.         this.$store.commit('replace', { métier: this.$métier })
55.         // on sauvegarde la session
56.         this.$session().save(this.$nuxt.context)
57.       }
58.     }
59. }
```

Le principal problème ici semble être l'absence de la donnée **[this.$métier]**. Pour y remédier nous allons :

- inclure la classe **[Métier]** de l'exemple **[vuejs-22]**. Nous la mettrons dans le dossier **[api]** ;
- injecter une fonction **[$métier]** dans le contexte du client **[nuxt]** qui donnera accès à cette classe ;

Tout d'abord la copie de la classe **[Métier]** dans le dossier **[api]** :



Une fois la classe **[Métier]** présente dans le projet, on crée un nouveau plugin pour le client **[nuxt]**. Ce plugin appelé **[pluginMétier]** va injecter une fonction **[$métier]** qui donnera accès à la classe **[Métier]** :

```
1.  /* eslint-disable no-console */
2.  // on crée un point d'accès à la couche [métier]
3.  import Métier from '@/api/client/Métier'
4.  export default (context, inject) => {
5.    // instanciation de la couche [métier]
6.    const métier = new Métier()
7.    // injection d'une fonction [$métier] dans le contexte
8.    inject('métier', () => métier)
9.    // log
10.   console.log('[fonction client $métier créée]')
11. }
```

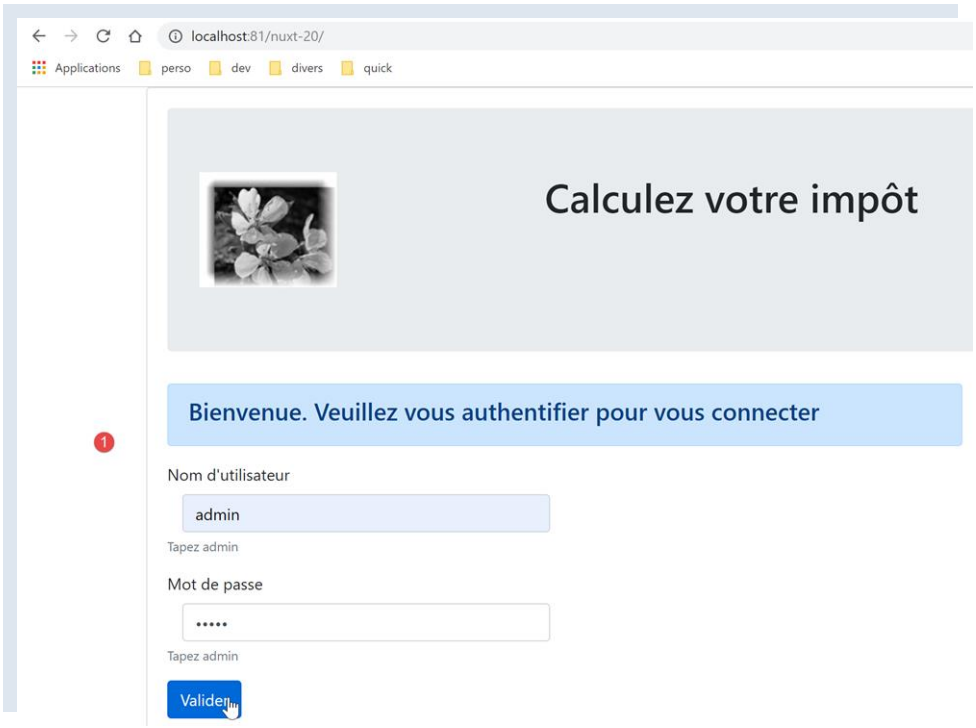
Ceci fait, nous pouvons corriger la page **[index]** :

```
1.  // cycle de vie : le composant vient d'être créé
2.    mounted() {
3.      // eslint-disable-next-line
4.      console.log("Authentification mounted");
5.      // l'utilisateur peut-il faire des simulations ?
6.      if (this.$store.state.started && this.$store.state.authenticated && this.$métier().taxAdminData) {
7.        // alors l'utilisateur peut faire des simulations
8.        this.$router.push({ name: 'calcul-impot' })
9.        // retour à la boucle événementielle
10.       return
11.     }
12.     // si la session jSON a déjà été démarrée, on ne la redémarre pas de nouveau
13.     ...
14.   },
15.
16.   // gestionnaires d'évts
17.   methods: {
18.     // ---------- authentification
19.     async login() {
20.       try {
21.         // début attente
22.         this.$emit('loading', true)
23.         // on n'est pas encore authentifié
24.         this.$store.commit('replace', { authenticated: false })
25.         // authentification bloquante auprès du serveur
26.         const response = await this.$dao().authentifierUtilisateur(this.user, this.password)
27.         // fin du chargement
28.         this.$emit('loading', false)
29.         // analyse de la réponse du serveur
30.         if (response.état !== 200) {
31.           // on affiche l'erreur
32.           this.message = response.réponse
33.           this.showError = true
34.           // retour à la boucle événementielle
35.           return
36.         }
37.         // pas d'erreur
38.         this.showError = false
39.         // on est authentifié
40.         this.$store.commit('replace', { authenticated: true })
41.         // --------- on demande maintenant les données de l'administration fiscale
42.         // au départ, pas de donnée
43.         this.$métier().setTaxAdminData(null)
44.         // début attente
45.         this.$emit('loading', true)
46.         // demande bloquante auprès du serveur
47.         const response2 = await this.$dao().getAdminData()
```

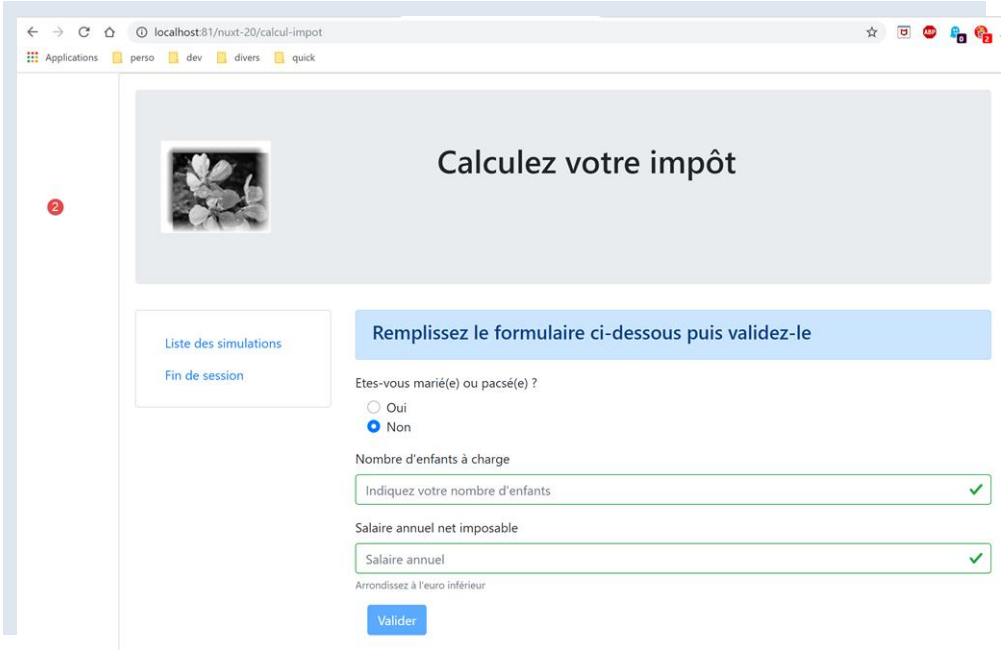
```
48.         // fin du chargement
49.         this.$emit('loading', false)
50.         // analyse de la réponse
51.         if (response2.état !== 1000) {
52.           // on affiche l'erreur
53.           this.message = response2.réponse
54.           this.showError = true
55.           // retour à la boucle événementielle
56.           return
57.         }
58.         // pas d'erreur
59.         this.showError = false
60.         // on mémorise dans la couche [métier] la donnée reçue
61.         this.$métier().setTaxAdminData(response2.réponse)
62.         // on peut passer au calcul de l'impôt
63.         this.$router.push({ name: 'calcul-impot' })
64.       } catch (error) {
65.         // on remonte l'erreur au composant principal
66.         this.$emit('error', error)
67.       } finally {
68.         // maj store
69.         this.$store.commit('replace', { métier: this.$métier() })
70.         // on sauvegarde la session
71.         this.$session().save(this.$nuxt.context)
72.       }
73.     }
74. }
```

- lignes 43, 61, 69, **[this.$métier]** a été remplacé par **[this.$métier()]** ;
- lignes 8, 63 : le nom de la page **[CalculImpot]** du projet **[vuejs-22]** est devenue la page **[calcul-impot]** dans le projet **[nuxt-20]** ;

Ces corrections faites, on peut tenter de valider la page d'authentification :



La page obtenue est la suivante :



On a bien obtenu la page de calcul de l'impôt. Maintenant regardons les logs :

En **[2]**, on voit que le fait d'être authentifié a été correctement mémorisé. En **[3-4]**, on voit qu'on a récupéré la donnée **[taxAdminData]** qui permet le calcul de l'impôt par la classe **[Métier]**.

4.17.5 étape 4

Examinons la page **[calcul-impot]** que nous avons obtenue :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <Layout :left="true" :right="true">
5.        <!-- formulaire de calcul de l'impôt à droite -->
6.        <FormCalculImpot slot="right" @resultatObtenu="handleResultatObtenu" />
7.        <!-- menu de navigation à gauche -->
8.        <Menu slot="left" :options="options" />
9.      </Layout>
10.     <!-- zone d'affichage des résultat du calcul de l'impôt sous le formulaire -->
11.     <b-row v-if="résultatObtenu" class="mt-3">
12.       <!-- zone de trois colonnes vide -->
13.       <b-col sm="3" />
14.       <!-- zone de neuf colonnes -->
15.       <b-col sm="9">
16.         <b-alert show variant="success">
17.           <span v-html="résultat"></span>
18.         </b-alert>
19.       </b-col>
20.     </b-row>
21.   </div>
22. </template>
```

```
23.
24. <script>
25. // imports
26. import FormCalculImpot from '@/components/form-calcul-impot'
27. import Menu from '@/components/menu'
28. import Layout from '@/components/layout'
29.
30. export default {
31.   // composants utilisés
32.   components: {
33.     Layout,
34.     FormCalculImpot,
35.     Menu
36.   },
37.   // état interne
38.   data() {
39.     return {
40.       // options du menu
41.       options: [
42.         {
43.           text: 'Liste des simulations',
44.           path: '/liste-des-simulations'
45.         },
46.         {
47.           text: 'Fin de session',
48.           path: '/fin-session'
49.         }
50.       ],
51.       // résultat du calcul de l'impôt
52.       résultat: '',
53.       résultatObtenu: false
54.     }
55.   },
56.   // cycle de vie
57.   created() {
58.     // eslint-disable-next-line
59.     console.log("CalculImpot created");
60.   },
61.   // méthodes de gestion des évts
62.   methods: {
63.     // résultat du calcul de l'impôt
64.     handleResultatObtenu(résultat) {
65.       // on construit le résultat en chaîne HTML
66.       const impôt = "Montant de l'impôt : " + résultat.impôt + ' euro(s)'
67.       const décôte = 'Décôte : ' + résultat.décôte + ' euro(s)'
68.       const réduction = 'Réduction : ' + résultat.réduction + ' euro(s)'
69.       const surcôte = 'Surcôte : ' + résultat.surcôte + ' euro(s)'
70.       const taux = "Taux d'imposition : " + résultat.taux
71.       this.résultat = impôt + '<br/>' + décôte + '<br/>' + réduction + '<br/>' + surcôte + '<br/>' + taux
72.       // affichage du résultat
73.       this.résultatObtenu = true
74.       // ---- maj du store [Vuex]
75.       // une simulation de +
76.       this.$store.commit('addSimulation', résultat)
77.       // on sauvegarde la session
78.       this.$session.save()
79.     }
80.   }
81. }
82. </script>
```

- lignes 44 et 48 : les liens du menu de navigation sont corrects. La page **[/fin-session]** n'existe pas. Le projet **[vuejs-22]** réglait ce problème avec du routage. Nous ferons de même avec le projet **[nuxt-20]** ;
- ligne 76 : on référence une mutation **[addSimulation]** qui n'existe pas pour l'instant. Nous allons la créer ;
- ligne 78 : comme dans la page **[index]**, il faut écrire **[this.$session().save(this.$nuxt.context)]** ;

Modifions le store **[store/index]**. Hérité du projet **[nuxt-12]**, il est pour l'instant le suivant :

```
1.  /* eslint-disable no-console */
2.
3.  // état du store
4.  export const state = () => ({
5.    // session jSON démarrée
6.    jsonSessionStarted: false,
7.    // utilisateur authentifié
8.    userAuthenticated: false,
```

```
9.    // cookie de session PHP
10.   phpSessionCookie: '',
11.   // adminData
12.   adminData: ''
13. })
14.
15. // mutations du store
16. export const mutations = {
17.   // remplacement du state
18.   replace(state, newState) {
19.     for (const attr in newState) {
20.       state[attr] = newState[attr]
21.     }
22.   },
23.   // reset du store
24.   reset() {
25.     this.commit('replace', { jsonSessionStarted: false, userAuthenticated: false, phpSessionCookie: '',
    adminData: '' })
26.   }
27. }
28.
29. // actions du store
30. export const actions = {
31.   nuxtServerInit(store, context) {
32.     // qui exécute ce code ?
33.     console.log('nuxtServerInit, client=', process.client, 'serveur=', process.server, 'env=', context.env)
34.     // init session
35.     initStore(store, context)
36.   }
37. }
38.
39. function initStore(store, context) {
40.   // store est le store à initialiser
41.   // on récupère la session
42.   const session = context.app.$session()
43.   // la session a-t-elle été déjà initialisée ?
44.   if (!session.value.initStoreDone) {
45.     // on démarre un nouveau store
46.     console.log("nuxtServerInit, initialisation d'un nouveau store")
47.     // on met le store dans la session
48.     session.value.store = store.state
49.     // le store est désormais initialisé
50.     session.value.initStoreDone = true
51.   } else {
52.     console.log("nuxtServerInit, reprise d'un store existant")
53.     // on met à jour le store avec le store de la session
54.     store.commit('replace', session.value.store)
55.   }
56.   // on sauvegarde la session
57.   session.save(context)
58.   // log
59.   console.log('initStore terminé, store=', store.state)
60. }
```

- lignes 3-27 : nous allons reprendre le state et les mutations de l'application **[vuejs-22]** (cf lien) :

```
1.  // état du store
2.  export const state = () => ({
3.    // session jSON démarrée
4.    started: false,
5.    // utilisateur authentifié
6.    authenticated: false,
7.    // cookie de session PHP
8.    phpSessionCookie: '',
9.    // liste des simulations
10.   simulations: [],
11.   // le n° de la dernière simulation
12.   idSimulation: 0,
13.   // couche [métier]
14.   métier: null
15. })
16.
17. // mutations du store
18. export const mutations = {
19.   // remplacement du state
20.   replace(state, newState) {
21.     for (const attr in newState) {
```

```
22.       state[attr] = newState[attr]
23.     }
24.   },
25.   // reset du store
26.   reset() {
27.         this.commit('replace', { started: false, authenticated: false, phpSessionCookie: '', idSimulation:
    0, simulations: [], métier: null })
28.   },
29.   // suppression ligne n° index
30.   deleteSimulation(state, index) {
31.     // eslint-disable-next-line no-console
32.     console.log('mutation deleteSimulation')
33.     // on supprime la ligne n° [index]
34.     state.simulations.splice(index, 1)
35.     console.log('store simulations', state.simulations)
36.   },
37.   // ajout d'une simulation
38.   addSimulation(state, simulation) {
39.     // eslint-disable-next-line no-console
40.     console.log('mutation addSimulation')
41.     // n° de la simulation
42.     state.idSimulation++
43.     simulation.id = state.idSimulation
44.     // on ajoute la simulation au tableau des simulations
45.     state.simulations.push(simulation)
46.   }
47. }
```

- lignes 4 et 6 : nous introduisons les propriétés déjà utilisées ;
- ligne 8 : nous gardons le cookie de session PHP. Il est fondamental pour que le client et le serveur **[nuxt]** aient la même session PHP avec le serveur de calcul de l'impôt ;
- ligne 10 : la liste des simulations faites par l'utilisateur ;
- ligne 12 : le n° de la dernière simulation faite par l'utiliasteur ;
- ligne 14 : la couche **[métier]** ;
- lignes 30-47 : les mutations présentes dans le store du projet **[vuejs-22]** et référencées par les pages de l'application. Le projet **[vuejs-22]** avait une mutation appelée **[clear]** qui vidait la liste des simulations. On ne la met pas car la mutation **[reset]** déjà présente devrait faire l'affaire ;
- lignes 26-28 : la mutation **[reset]** est modifiée pour prendre en compte le nouveau contenu du state ;

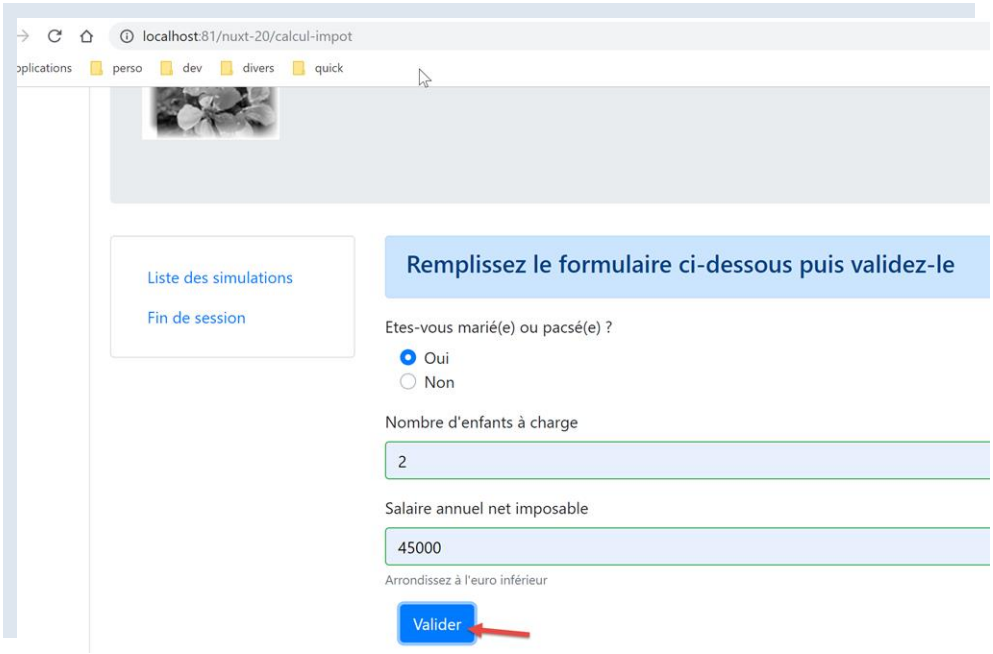
La page **[calcul-impot]** utilise le composant **[form-calcul-impot]** suivant :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <!-- formulaire HTML -->
4.    <b-form @submit.prevent="calculerImpot" class="mb-3">
5.      <!-- message sur 12 colonnes sur fond bleu -->
6.      <b-row>
7.        <b-col sm="12">
8.          <b-alert show variant="primary">
9.            <h4>Remplissez le formulaire ci-dessous puis validez-le</h4>
10.         </b-alert>
11.       </b-col>
12.     </b-row>
13.     <!-- éléments du formulaire -->
14.     <!-- première ligne -->
15.     <b-form-group label="Etes-vous marié(e) ou pacsé(e) ?">
16.       <!-- boutons radio sur 5 colonnes-->
17.       <b-col sm="5">
18.         <b-form-radio v-model="marié" value="oui">Oui</b-form-radio>
19.         <b-form-radio v-model="marié" value="non">Non</b-form-radio>
20.       </b-col>
21.     </b-form-group>
22.     <!-- deuxième ligne -->
23.     <b-form-group label="Nombre d'enfants à charge" label-for="enfants">
24.       <b-form-input id="enfants" v-model="enfants" :state="enfantsValide" type="text" placeholder="Indiquez
    votre nombre d'enfants"></b-form-input>
25.       <!-- message d'erreur éventuel -->
26.       <b-form-invalid-feedback :state="enfantsValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
27.     </b-form-group>
28.     <!-- troisème ligne -->
29.     <b-form-group label="Salaire annuel net imposable" label-for="salaire" description="Arrondissez à
    l'euro inférieur">
```

```
30.       <b-form-input id="salaire" v-model="salaire" :state="salaireValide" type="text" placeholder="Salaire
    annuel"></b-form-input>
31.       <!-- message d'erreur éventuel -->
32.       <b-form-invalid-feedback :state="salaireValide">Vous devez saisir un nombre positif ou nul</b-form-
    invalid-feedback>
33.     </b-form-group>
34.     <!-- quatrième ligne, bouton [submit] -->
35.     <b-col sm="3">
36.       <b-button :disabled="formInvalide" type="submit" variant="primary">Valider</b-button>
37.     </b-col>
38.   </b-form>
39. </template>
40.
41. <!-- script -->
42. <script>
43. export default {
44.   // état interne
45.   data() {
46.     return {
47.       // marié ou pas
48.       marié: 'non',
49.       // nombre d'enfants
50.       enfants: '',
51.       // salaire annuel
52.       salaire: ''
53.     }
54.   },
55.   // état interne calculé
56.   computed: {
57.     // validation du formulaire
58.     formInvalide() {
59.       return (
60.         // salaire invalide
61.         !this.salaire.match(/^\s*\d+\s*$/) ||
62.         // ou enfants invalide
63.         !this.enfants.match(/^\s*\d+\s*$/) ||
64.         // ou données fiscales pas obtenues
65.         !this.$métier.taxAdminData
66.       )
67.     },
68.     // validation du salaire
69.     salaireValide() {
70.       // doit être numérique >=0
71.       return Boolean(this.salaire.match(/^\s*\d+\s*$/) || this.salaire.match(/^\s*$/))
72.     },
73.     // validation des enfants
74.     enfantsValide() {
75.       // doit être numérique >=0
76.       return Boolean(this.enfants.match(/^\s*\d+\s*$/) || this.enfants.match(/^\s*$/))
77.     }
78.   },
79.   // cycle de vie
80.   created() {
81.     // log
82.     // eslint-disable-next-line
83.     console.log("FormCalculImpot created");
84.   },
85.   // gestionnaire d'évts
86.   methods: {
87.     calculerImpot() {
88.       // on calcule l'impôt à l'aide de la couche [métier]
89.       const résultat = this.$métier.calculerImpot(this.marié, Number(this.enfants), Number(this.salaire))
90.       // eslint-disable-next-line
91.       console.log("résultat=", résultat);
92.       // on complète le résultat
93.       résultat.marié = this.marié
94.       résultat.enfants = this.enfants
95.       résultat.salaire = this.salaire
96.       // on émet l'évt [resultatObtenu]
97.       this.$emit('resultatObtenu', résultat)
98.     }
99.   }
100.}
101.</script>
```

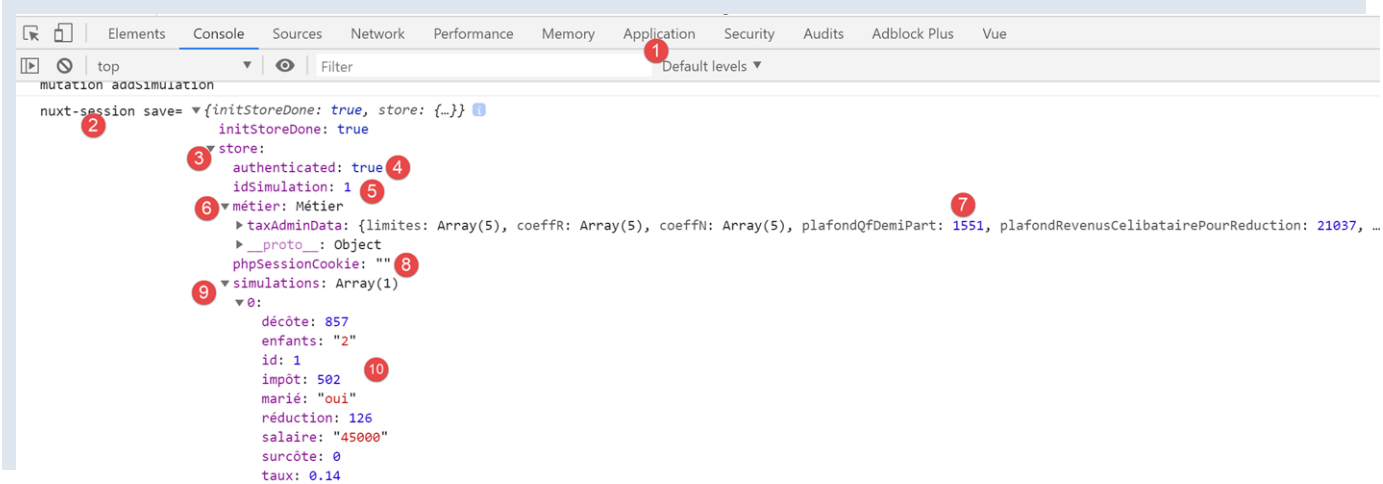
- lignes 65, 89 : la référence **[this.$métier]** doit être changée en **[this.$métier()]** ;

Ces corrections faites, on peut tenter une simulation :



On obtient la réponse suivante :

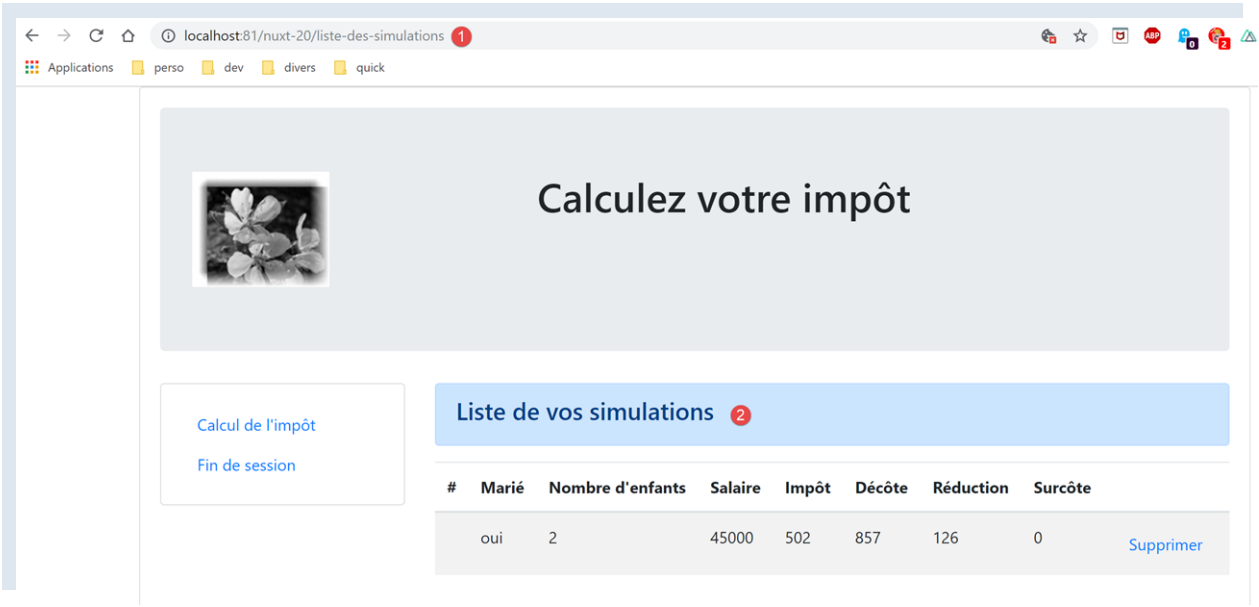
Si on regarde les logs :



- en **[9-10]**, on voit que la 1ère simulation se trouve bien dans le **[store]** ;
- en **[5]**, le n° de la dernière simulation a bien été incrémenté ;

4.17.6 étape 5

Maintenant que nous avons fait une simulation, cliquons sur le lien **[Liste des simulations]**. Nous obtenons la page suivante :



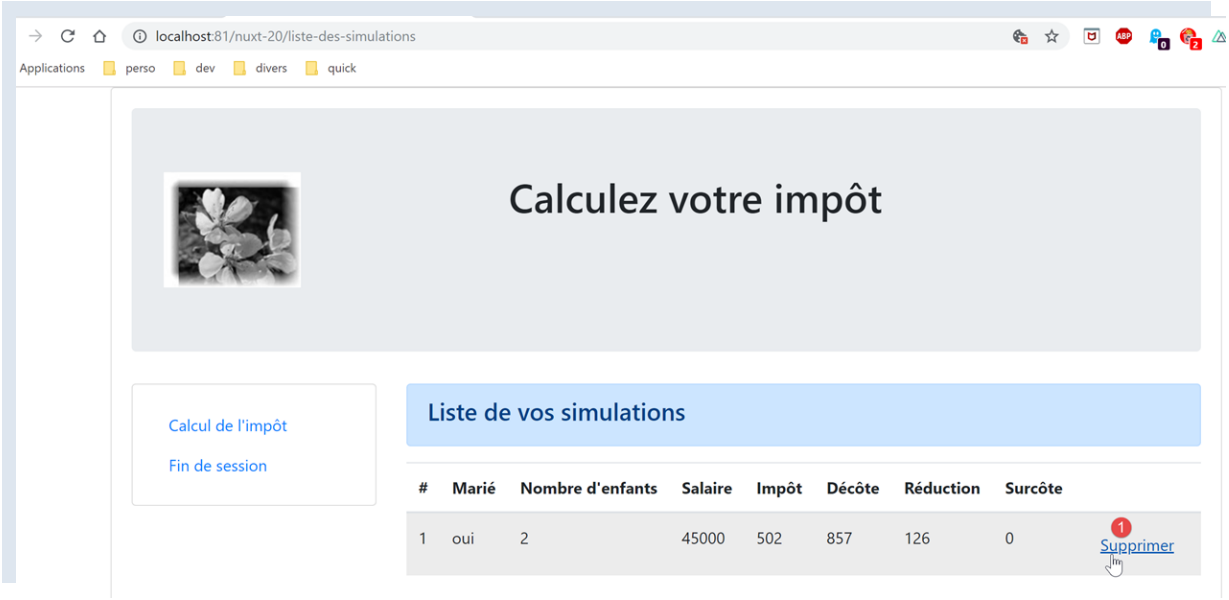
Le routage du client **[nuxt]** s'est fait correctement. Regardons le code de la page **[liste-des-simulations]** :

```
1.  <!-- définition HTML de la vue -->
2.  <template>
3.    <div>
4.      <!-- mise en page -->
5.      <Layout :left="true" :right="true">
6.        <!-- simulations dans colonne de droite -->
7.        <template slot="right">
8.          <template v-if="simulations.length == 0">
9.            <!-- pas de simulations -->
10.           <b-alert show variant="primary">
11.             <h4>Votre liste de simulations est vide</h4>
12.           </b-alert>
13.         </template>
14.         <template v-if="simulations.length != 0">
15.           <!-- il y a des simulations -->
16.           <b-alert show variant="primary">
17.             <h4>Liste de vos simulations</h4>
18.           </b-alert>
19.           <!-- tableau des simulations -->
20.           <b-table :items="simulations" :fields="fields" striped hover responsive>
21.             <template v-slot:cell(action)="data">
22.               <b-button @click="supprimerSimulation(data.index)" variant="link">Supprimer</b-button>
23.             </template>
24.           </b-table>
25.         </template>
26.       </template>
27.       <!-- menu de navigation dans colonne de gauche -->
28.       <Menu slot="left" :options="options" />
29.     </Layout>
30.   </div>
31. </template>
32.
33. <script>
34. // imports
35. import Layout from '@/components/layout'
36. import Menu from '@/components/menu'
37. export default {
```

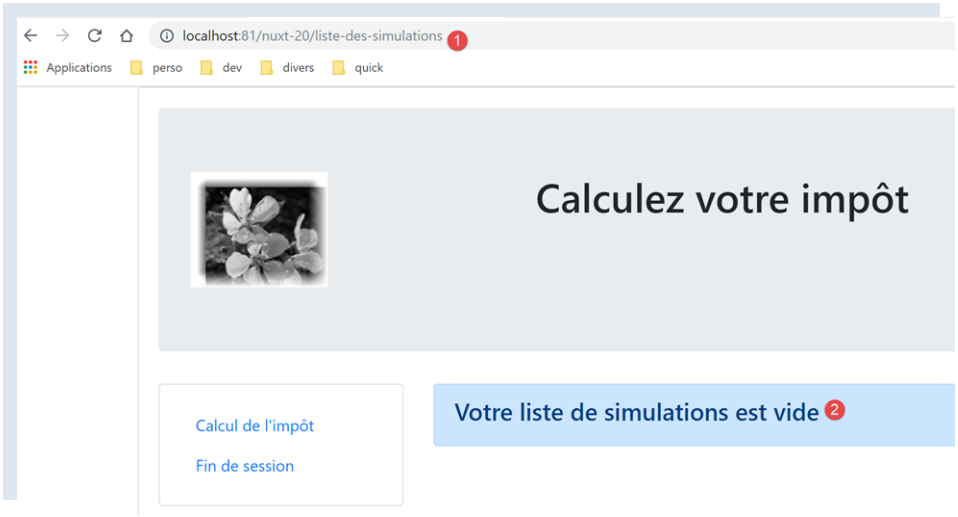
```
38.   // composants
39.   components: {
40.     Layout,
41.     Menu
42.   },
43.   // état interne
44.   data() {
45.     return {
46.       // options du menu de navigation
47.       options: [
48.         {
49.           text: "Calcul de l'impôt",
50.           path: '/calcul-impot'
51.         },
52.         {
53.           text: 'Fin de session',
54.           path: '/fin-session'
55.         }
56.       ],
57.       // paramètres de la table HTML
58.       fields: [
59.         { label: '#', key: 'id' },
60.         { label: 'Marié', key: 'marié' },
61.         { label: "Nombre d'enfants", key: 'enfants' },
62.         { label: 'Salaire', key: 'salaire' },
63.         { label: 'Impôt', key: 'impôt' },
64.         { label: 'Décôte', key: 'décôte' },
65.         { label: 'Réduction', key: 'réduction' },
66.         { label: 'Surcôte', key: 'surcôte' },
67.         { label: '', key: 'action' }
68.       ]
69.     }
70.   },
71.   // état interne calculé
72.   computed: {
73.     // liste des simulations prise dans le store Vuex
74.     simulations() {
75.       return this.$store.state.simulations
76.     }
77.   },
78.   // cycle de vie
79.   created() {
80.     // eslint-disable-next-line
81.     console.log("ListeSimulations created");
82.   },
83.   // méthodes
84.   methods: {
85.     supprimerSimulation(index) {
86.       // eslint-disable-next-line
87.       console.log("supprimerSimulation", index);
88.       // suppression de la simulation n° [index]
89.       this.$store.commit('deleteSimulation', index)
90.       // on sauvegarde la session
91.       this.$session.save()
92.     }
93.   }
94. }
95. </script>
```

- lignes 47-56 : les cibles du menu de navigation sont correctes ;
- ligne 75 : le store est correctement référencé ;
- ligne 89 : on utilise une mutation **[deleteSimulation]** que nous avons intégrée lors de l'étape précédente ;
- ligne 91 : cette ligne doit être réécrite comme **[this.$session().save(this.$nuxt.context)]** ;

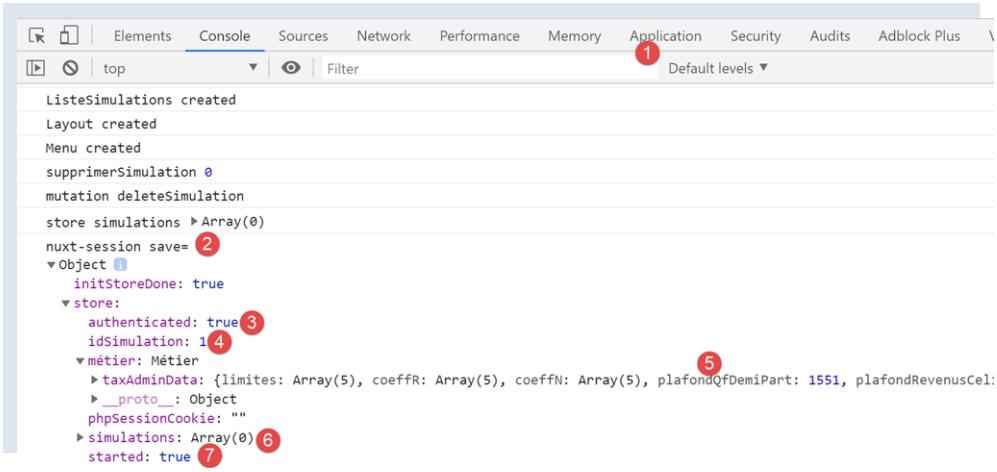
Nous faisons les modifications nécessaires, puis nous tentons de supprimer la simulation affichée :



On obtient alors la page suivante :

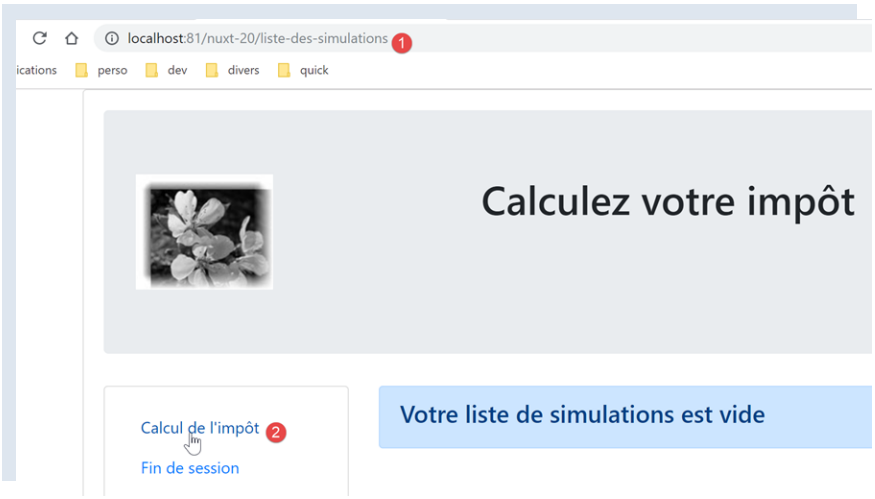


Regardons les logs :

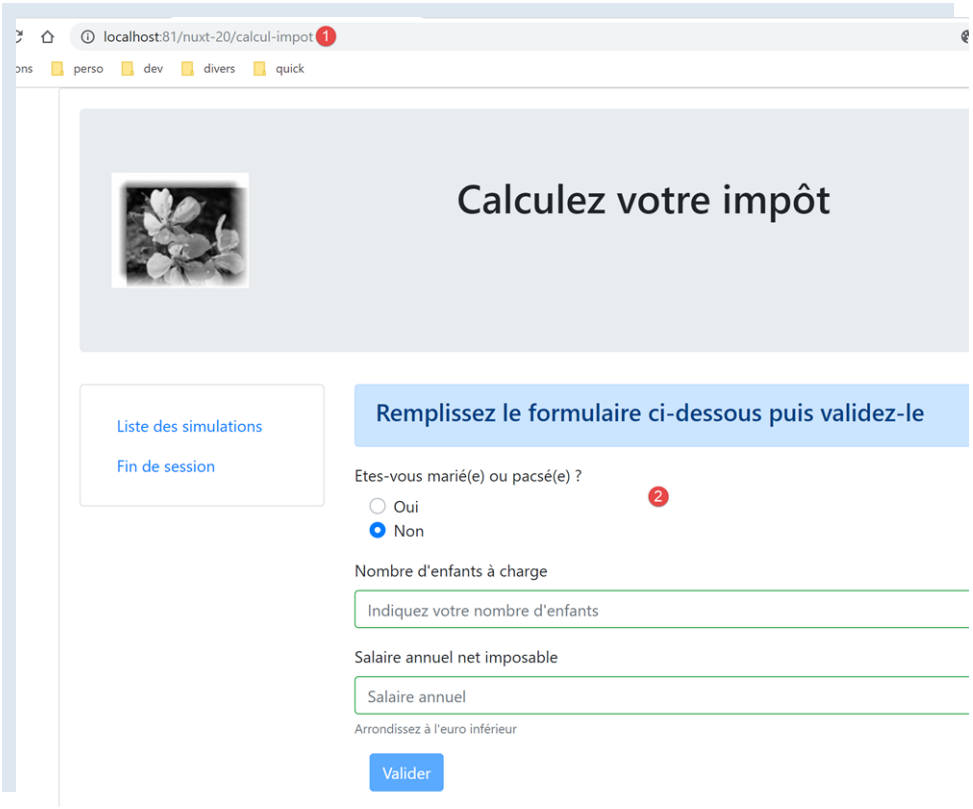


- en **[6]**, on voit que le tableau des simulations est vide ;

Maintenant revenons au formulaire de calcul de l'impôt :



On obtient la page suivante :



Donc le routage a fonctionné.

4.17.7 étape 6

Il nous reste à gérer l'option de navigation **`[Fin de session]`** du menu de navigation :

```
1.  // options du menu
2.        options: [
3.          {
4.            text: 'Liste des simulations',
5.            path: '/liste-des-simulations'
6.          },
7.          {
8.            text: 'Fin de session',
9.            path: '/fin-session'
```

```
10.         }
11. ]
```

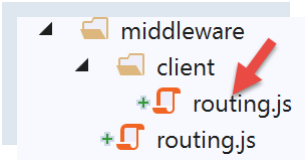
- ligne 9, la page **[/fin-session]** n'existe pas. Le projet **[vuejs-22]** gérait ce cas avec des règles de routage dans un fichier **[router.js]**:

```
1.  // imports
2.  import Vue from 'vue'
3.  import VueRouter from 'vue-router'
4.  // les vues
5.  import Authentification from './views/Authentification'
6.  import CalculImpot from './views/CalculImpot'
7.  import ListeSimulations from './views/ListeSimulations'
8.  import NotFound from './views/NotFound'
9.  // la session
10. import session from './session'
11.
12. // plugin de routage
13. Vue.use(VueRouter)
14.
15. // les routes de l'application
16. const routes = [
17.   // authentification
18.   { path: '/', name: 'authentification', component: Authentification },
19.   { path: '/authentification', name: 'authentification2', component: Authentification },
20.   // calcul de l'impôt
21.   {
22.     path: '/calcul-impot', name: 'calculImpot', component: CalculImpot,
23.     meta: { authenticated: true }
24.   },
25.   // liste des simulations
26.   {
27.     path: '/liste-des-simulations', name: 'listeSimulations', component: ListeSimulations,
28.     meta: { authenticated: true }
29.   },
30.   // fin de session
31.   {
32.     path: '/fin-session', name: 'finSession'
33.   },
34.   // page inconnue
35.   {
36.     path: '*', name: 'notFound', component: NotFound,
37.   },
38. ]
39.
40. // le routeur
41. const router = new VueRouter({
42.   // les routes
43.   routes,
44.   // le mode d'affichage des URL
45.   mode: 'history',
46.   // l'URL de base de l'application
47.   base: '/client-vuejs-impot/'
48. })
49.
50. // vérification des routes
51. router.beforeEach((to, from, next) => {
52.   // eslint-disable-next-line no-console
53.   console.log("router to=", to, "from=", from);
54.   // route réservée aux utilisateurs authentifiés ?
55.   if (to.meta.authenticated && !session.authenticated) {
56.     next({
57.       // on passe à l'authentification
58.       name: 'authentification',
59.     })
60.     // retour à la boucle événementielle
61.     return;
62.   }
63.   // cas particulier de la fin de session
64.   if (to.name === "finSession") {
65.     // on nettoie la session
66.     session.clear();
67.     // on va sur la vue [authentification]
68.     next({
69.       name: 'authentification',
```

```
70.     })
71.     // retour à la boucle événementielle
72.     return;
73.   }
74.   // autres cas - vue suivante normale du routage
75.   next();
76. })
77.
78. // export du router
79. export default router
```

- les lignes 64-76 géraient le cas particulier de la route vers le chemin **[/fin-session]** ;
- ligne 66 : on vide la session courante ;
- lignes 68-70 : on affiche la vue **[authentification]** ;

Nous allons essayer de faire quelque chose d'analogue dans le fichier de routage du client **[nuxt]** :



Le script **[client/routing.js]** devient le suivant :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware client], process.server', process.server, ', process.client=', process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session nuxt
7.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18.
19.   // où va-t-on ?
20.   const to = context.route.path
21.   if (to === '/fin-session') {
22.     // on nettoie la session
23.     const session = context.app.$session()
24.     session.reset(context)
25.     // on redirige vers la page index
26.     context.redirect({ name: 'index' })
27.   }
```

- on a rajouté les lignes **[19-27]** au code existant ;
- ligne 20 : on récupère le **[path]** de la cible de la route courante ;
- ligne 21 : on regarde si c'est **[/fin-session]**. Si oui :
  - lignes 23-24 : la session est réinitialisée ;
  - ligne 26 : on redirige le client **[nuxt]** vers la page d'accueil ;

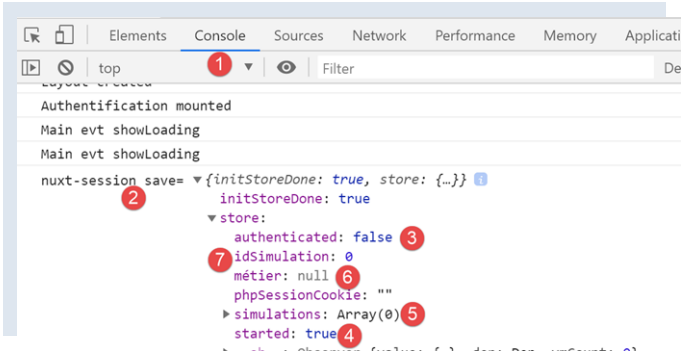
La méthode **[session.reset(context)]** (ligne 24) de la session est la suivante :

```
1.    // reset de la session
2.    reset(context) {
3.      console.log('nuxt-session reset')
4.      // reset du store
5.      context.store.commit('reset')
6.      // sauvegarde du nouveau store en session et sauvegarde de la session
7.      this.save(context)
8.    }
```

La méthode **[context.store.commit('reset')]** (ligne 5) est la suivante :

```
1.    // reset du store
2.    reset() {
3.      this.commit('replace', { started: false, authenticated: false, phpSessionCookie: '', idSimulation: 0, s
    imulations: [], métier: null })
4.    },
```

Lorsqu'on utilise maintenant le lien **[Fin de session]**, la page d'accueil est affichée avec les logs suivants :



- en **[3]**, on voit qu'on n'est plus authentifié ;
- en **[4]**, on voit que la session jSON est démarrée ;
- en **[6]**, la couche **[métier]** n'est plus présente dans le store (elle est toujours présente dans les pages avec **[this.$métier()]**) ;
- en **[5, 7]**, il n'y a plus de simulations ;

Il faut bien comprendre ce qui se passe dans une fin de session :

- la session **[nuxt]** est réinitialisée : la propriété **[started]** du store passe à **[false]** ;
- il y a redirection vers la page **[index]** ;
- la méthode **[mounted]** de la page **[index]** est exécutée. Celle-ci démarre une nouvelle session jSON avec le serveur de calcul de l'impôt. Si l'opération réussit, la propriété **[started]** du store passe à **[true]** ;

4.17.8 étape 7

A ce stade, l'application **[nuxt-20]** a toutes les fonctionnalités de l'application **[vuejs-22]**. Le portage semble terminé.

Nous allons aller un peu plus loin dans un esprit **[nuxt]**. La méthode **[mounted]** de la page **[index]** pose problème. Elle lance une opération asynchrone dont un moteur de recherche n'attendra pas la fin. On sait que dans ce cas là, il faut mettre l'opération asynchrone dans une fonction **[asyncData]** car alors, le serveur **[nuxt]** qui l'exécute attend qu'elle soit terminée avant de délivrer la page au moteur de recherche.

Nous nous aidons ici de la fonction **[asyncData]** écrite dans l'application **[nuxt-13]** pour la page **[index]** :

```
1.  export default {
2.    name: 'InitSession',
3.    // composants utilisés
4.    components: {
5.      Layout,
6.      Navigation
7.    },
8.    // données asynchrones
9.    async asyncData(context) {
10.     // log
11.     console.log('[index asyncData started]')
12.     // on ne fait pas les choses deux fois si la page a déjà été demandée
13.     if (process.server && context.store.state.jsonSessionStarted) {
14.       console.log('[index asyncData canceled]')
15.       return { result: '[succès]' }
16.     }
17.     try {
18.       // on démarre une session jSON
19.       const dao = context.app.$dao()
20.       const response = await dao.initSession()
21.       // log
22.       console.log('[index asyncData response=]', response)
```

```
23.       // on récupère le cookie de session PHP pour les prochaines requêtes
24.       const phpSessionCookie = dao.getPhpSessionCookie()
25.       // on mémorise le cookie de session PHP dans la session [nuxt]
26.       context.store.commit('replace', { phpSessionCookie })
27.       // y-a-t-il eu erreur ?
28.       if (response.état !== 700) {
29.         // l'erreur se trouve dans response.réponse
30.         throw new Error(response.réponse)
31.       }
32.       // on note le fait que la session jSON a démarré
33.       context.store.commit('replace', { jsonSessionStarted: true })
34.       // on rend le résultat
35.       return { result: '[succès]' }
36.     } catch (e) {
37.       // log
38.       console.log('[index asyncData error=]', e)
39.       // on note le fait que la session jSON n'a pas démarré
40.       context.store.commit('replace', { jsonSessionStarted: false })
41.       // on signale l'erreur
42.       return { result: '[échec]', showErrorLoading: true, errorLoadingMessage: e.message }
43.     } finally {
44.       // on sauvegarde le store
45.       const session = context.app.$session()
46.       session.save(context)
47.       // log
48.       console.log('[index asyncData finished]')
49.     }
50.   },
51.   // cycle de vie
52.   beforeCreate() {
53.     console.log('[index beforeCreate]')
54.   },
55.   created() {
56.     console.log('[index created]')
57.   },
58.   beforeMount() {
59.     console.log('[index beforeMount]')
60.   },
61.   mounted() {
62.     console.log('[index mounted]')
63.     // client seulement
64.     if (this.showErrorLoading) {
65.       console.log('[index mounted, showErrorLoading=true]')
66.       this.$eventBus().$emit('errorLoading', true, this.errorLoadingMessage)
67.     }
68.   }
```

- lignes 13, 33, 40 : il faut changer la propriété **[jsonSessionStarted]** en **[started]** ;
- ligne 13 : dans l'application **[nuxt-12]**, seul le serveur **[nuxt]** exécutait la page **[index]** et sa fonction **[asyncData]**. Le client **[nuxt]** n'exécutait la page **[index]** qu'après l'avoir reçue du serveur **[nuxt]** et alors il n'exécutait pas la fonction **[asyncData]**. Dans **[nuxt-20]**, c'est différent : le lien **[Fin de session]** va afficher la page **[index]** dans l'environnement du client **[nuxt]**. La fonction **[asyncData]** va alors être exécutée. Cependant, lorsqu'on arrive à la page **[index]** de cette façon, la session **[nuxt]** a été réinitialisée entre-temps et la propriété **[started]** du store vaut **[false]** et la condition de la ligne 13 sera forcément fausse. On peut donc laisser **[process.server]** et ainsi le client **[nuxt]** ne fera pas ce test ;
- lignes 15, 35, 42 : une propriété **[result]** est mise dans les propriétés **[data]** de la page **[index]**. Dans **[nuxt-20]**, cette propriété ne sera pas utilisée et nous l'enlèverons du résultat rendu par la fonction ;
- lignes 61-67 : cette méthode **[mounted]** doit être conservée car c'est elle qui permet au client **[nuxt]** d'afficher le message d'erreur. Néanmoins la façon de gérer l'erreur sera modfiée ;

Dans la page **[index]** actuelle, nous intégrons la fonction **[asyncData]** ci-dessus en lieu et place de l'ancienne fonction **[mounted]** et nous ajoutons une nouvelle fonction **[mounted]**. Le code de la page **[index]** de l'exemple **[nuxt-20]** devient alors le suivant :

```
1.  <!-- dynamique de la vue -->
2.  <script>
3.  /* eslint-disable no-console */
4.  import Layout from '@/components/layout'
5.  export default {
6.    // composants utilisés
7.    components: {
8.      Layout
9.    },
10.   // état du composant
11.   data() {
12.     return {
```

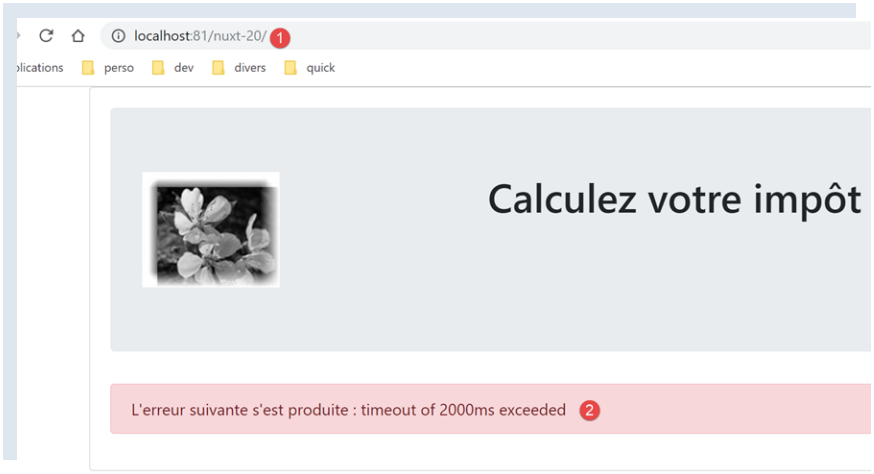
```
      // utilisateur
      user: '',
      // son mot de passe
      password: '',
      // affichage erreur
      showError: false
    }
  },

  // propriétés calculées
  computed: {
    // saisies valides
    valid() {
      return this.user && this.password && this.$store.state.started
    }
  },
  // données asynchrones
  async asyncData(context) {
    // log
    console.log('[index asyncData started]')
    // on ne fait pas les choses deux fois si la page a déjà été demandée
    if (process.server && context.store.state.started) {
      console.log('[index asyncData canceled]')
      return
    }
    try {
      // on démarre une session jSON
      const dao = context.app.$dao()
      const response = await dao.initSession()
      // log
      console.log('[index asyncData response=]', response)
      // on récupère le cookie de session PHP pour les prochaines requêtes
      const phpSessionCookie = dao.getPhpSessionCookie()
      // on mémorise le cookie de session PHP dans la session [nuxt]
      context.store.commit('replace', { phpSessionCookie })
      // y-a-t-il eu erreur ?
      if (response.état !== 700) {
        // l'erreur se trouve dans response.réponse
        throw new Error(response.réponse)
      }
      // on note le fait que la session jSON a démarré
      context.store.commit('replace', { started: true })
      // on rend le résultat
      return
    } catch (e) {
      // log
      console.log('[index asyncData error=]', e.message)
      // on note le fait que la session jSON n'a pas démarré
      context.store.commit('replace', { started: false })
      // on signale l'erreur
      return { showErrorLoading: true, errorLoadingMessage: e.message }
    } finally {
      // on sauvegarde le store
      const session = context.app.$session()
      session.save(context)
      // log
      console.log('[index asyncData finished]')
    }
  },
  // cycle de vie
  beforeCreate() {
    console.log('[index beforeCreate]')
  },
  created() {
    console.log('[index created]')
  },
  beforeMount() {
    // client seulement
    console.log('[index beforeMount]')
    // gestion de l'erreur éventuelle
    if (this.showErrorLoading) {
      // log
      console.log('[index beforeMount, showErrorLoading=true]')
      // on remonte l'erreur au composant principal
      this.$emit('error', new Error(this.errorLoadingMessage))
    }
  },
```

```
90.   mounted() {
91.     console.log('[index mounted]')
92.   },
93.
94.   // gestionnaires d'évts
95.   methods: {
96.     // ---------- authentification
97.     async login() {
98.       …
99.   }
100.}
101.</script>
```

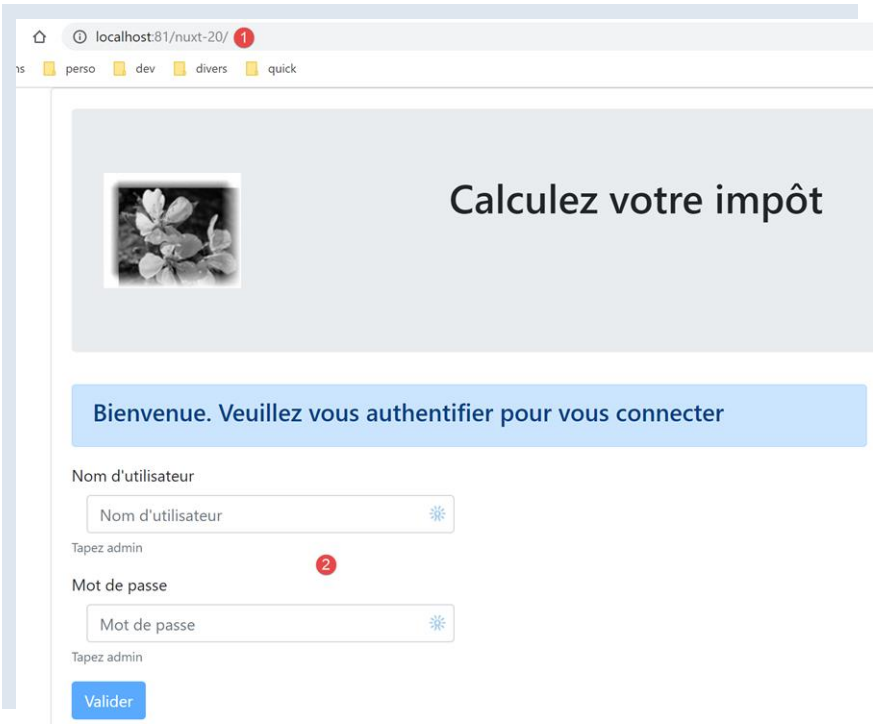
- lignes 56, 63 : la fonction **[asyncData]** ne rend plus la propriété **[result]** inutilisée ici ;
- ligne 79 : la méthode **[beforeMount]** du client **[nuxt]**. Elle a été préférée à la méthode **[mounted]** pour gérer l'erreur éventuelle de **[asyncData]** ;
- ligne 83 : on teste si la propriété **[errorLoading]** a été positionnée. Elle ne peut l'être que par la fonction **[asyncData]** ;
- lignes 83-88 : si fonction **[asyncData]** a signalé une erreur, on la passe à la page **[default]** via l'événement **[error]**. C'était comme cela que l'ancienne fonction **[created]** que nous venons de remplacer, gérait l'éventuelle erreur ;

Faisons quelques tests.

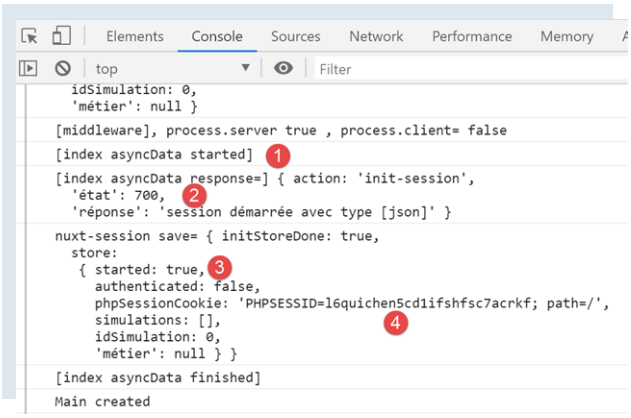
Nous supprimons d'abord et le cookie de la session **[nuxt]** et le cookie de session PHP s'ils existent. Nous demandons ensuite la page **[http://localhost:81/nuxt-20/]** alors que le serveur de calcul de l'impôt n'est pas lancé. Nous obtenons la page suivante :



Nous rechargeons la même page après avoir lancé le serveur de calcul de l'impôt :

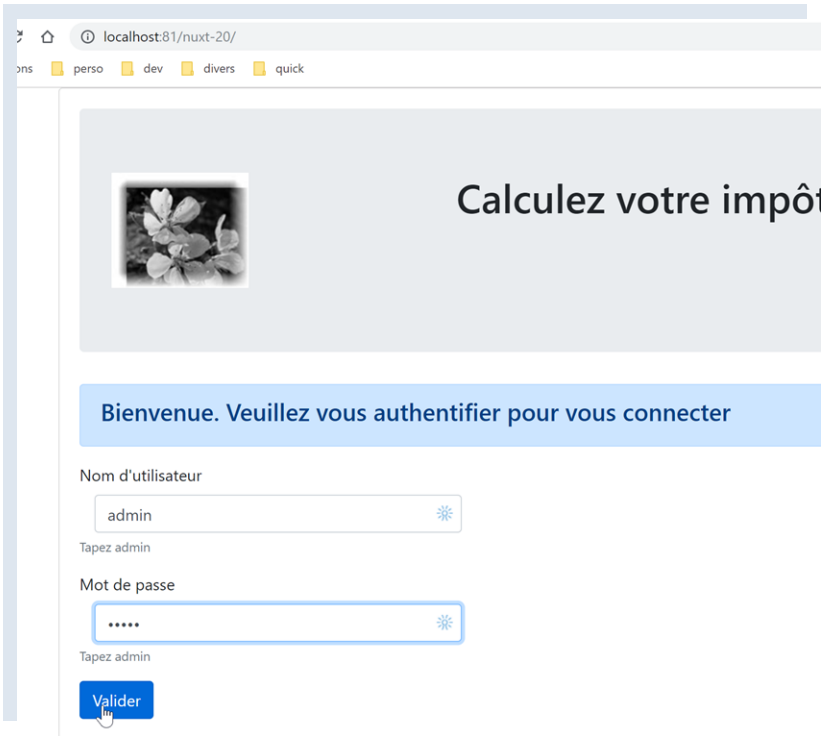


Regardons les logs :

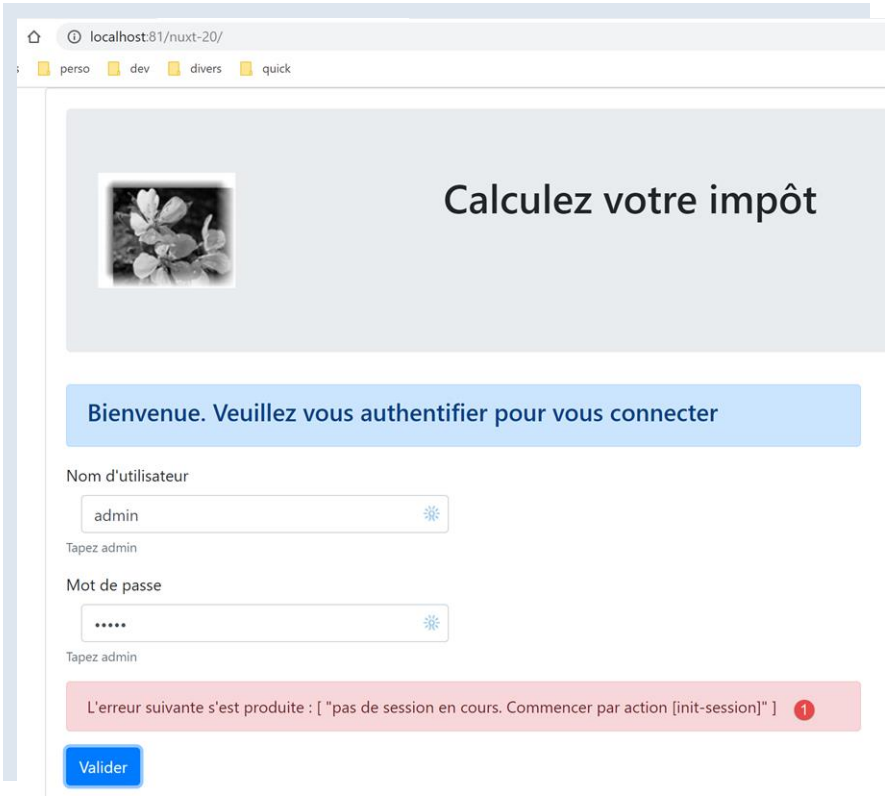


- en **[2-3]**, on voit que la session jSON a été démarrée ;
- en **[4]**, on voit le cookie de session PHP que le serveur **[nuxt]** a récupéré lors de son échange avec le serveur de calcul de l'impôt. Le client **[nuxt]** va désormais l'utiliser ;

Maintenant identifions-nous :



Nous obtenons la page suivante :



En **[1]**, nous avons obtenu un message d'erreur. Cela veut dire que le navigateur n'a pas envoyé le bon cookie de la session PHP démarrée par le serveur **[nuxt]** à l'étape précédente. Dans **[nuxt-13]**, le passage du cookie de session PHP du serveur **[nuxt]** au client **[nuxt]** se faisait dans le routage du client **[nuxt]** du script **[middleware/client/routing]** :

```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware client], process.server', process.server, ', process.client=', process.client)
5.    // gestion du cookie de la session PHP dans le navigateur
```

```
6.    // le cookie de la session PHP du navigateur doit être identique à celui trouvé en session nuxt
7.    // l'acion [fin-session] reçoit un nouveau cookie PHP (serveur comme client nuxt)
8.    // si c'est le serveur qui le reçoit, le client doit le transmettre au navigateur
9.    // pour ses propres échanges avec le serveur PHP
10.   // on est ici dans un routing client
11.
12.   // on récupère le cookie de la session PHP
13.   const phpSessionCookie = context.store.state.phpSessionCookie
14.   if (phpSessionCookie) {
15.     // s'il existe, on affecte le cookie de session PHP au navigateur
16.     document.cookie = phpSessionCookie
17.   }
18.
19.   // on met dans la session le nom de la page où on va - pas de redirection serveur
20.   context.store.commit('replace', { serverRedirection: false, from: context.route.name })
21.   // on sauvegarde le store dans la session [nuxt]
22.   const session = context.app.$session()
23.   session.value.store = context.store.state
24.   session.save(context)
25. }
```

Ce sont les lignes 13-17 qui permettent au client **[nuxt]** de récupérer le cookie de la session PHP du serveur **[nuxt]**.

Le problème ici c'est que lorsqu'on clique sur le bouton **[Valider]**, il n'y a pas de routage du client **[nuxt]**. Sa fonction de routage n'est alors pas appelée. On règle le problème en dupliquant les lignes 12-17 au début de la méthode d'authentification de la page **[index]** :

```
1.    // gestionnaires d'évts
2.    methods: {
3.      // ----------- authentification
4.      async login() {
5.        // on récupère le cookie de session PHP dans le store
6.        const phpSessionCookie = this.$store.state.phpSessionCookie
7.        if (phpSessionCookie) {
8.          // s'il existe, on affecte le cookie de session PHP au navigateur
9.          document.cookie = phpSessionCookie
10.       }
11.       try {
12.         // début attente
13.         this.$emit('loading', true)
14.         // on n'est pas encore authentifié
15.
```

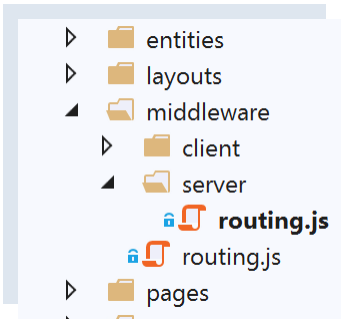
Lignes 5-10, on récupère dans le store le cookie de la session PHP initiée par le serveur **[nuxt]**. Cette modification faite, on récupère bien la page du calcul de l'impôt, signifiant que l'authentification a fonctionné.

4.17.9 étape 8

Nous avons une application fonctionnelle et qui travaille dans un esprit **[nuxt]**. Comme nous l'avons fait pour l'application **[nuxt-13]**, nous allons nous intéresser à la navigation du serveur **[nuxt]**. Comme il a déjà été dit, l'utilisateur n'est pas censé taper à la main les URL de l'application. Il est censé utiliser les liens qui lui sont présentés et qui sont exécutés par le client **[nuxt]** qui fonctionne alors en mode SPA. Néanmoins, on va faire en sorte que la navigation du serveur **[nuxt]** laisse toujours l'application dans un état stable.

De l'étude faite pour **[nuxt-13]** (cf paragraphe lien), nous savons qu'il faut :

- modifier le script **[midleware/routing]** ;
- ajouter un script **[middleware/server/routing]** ;



Le script **[middleware/routing]** est modifié de la façon suivante :

```
1.  /* eslint-disable no-console */
2.
3.  // on importe les middlewares du client et du server
4.  import clientRouting from './client/routing'
5.  import serverRouting from './server/routing'
6.  export default function(context) {
7.    // qui exécute ce code ?
8.    console.log('[middleware], process.server', process.server, ', process.client=', process.client)
9.    if (process.client) {
10.     // routage client
11.     clientRouting(context)
12.   } else {
13.     // routage serveur
14.     serverRouting(context)
15.   }
16. }
```

- ligne 5 : on importe le script de routage du serveur **[nuxt]** ;
- lignes 12-14 : si c'est le serveur **[nuxt]** qui exécute le code, on utilise sa fonction de routage ;

Le script **[middleware/server/routing]** est le suivant :

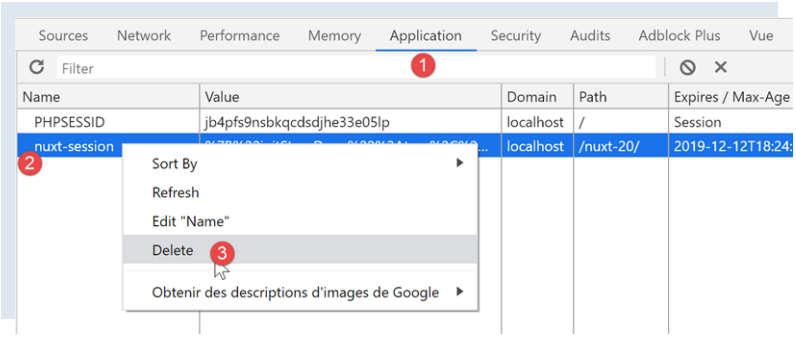
```
1.  /* eslint-disable no-console */
2.  export default function(context) {
3.    // qui exécute ce code ?
4.    console.log('[middleware server], process.server', process.server, ', process.client=', process.client)
5.
6.    // on récupère quelques informations ici et là
7.    const store = context.store
8.    // d'où vient-on ?
9.    const from = store.state.from || 'nowhere'
10.   // où va-t-on ?
11.   let to = context.route.name
12.
13.   // cas particulier de /fin-session qui n'a pas d'attribut [name]
14.   if (context.route.path === '/fin-session') {
15.     to = 'fin-session'
16.   }
17.   // éventuelle redirection
18.   let redirection = ''
19.   // gestion du routage terminé
20.   let done = false
21.
22.   // est-on déjà dans une redirection du serveur [nuxt]?
23.   if (store.state.serverRedirection) {
24.     // rien à faire
25.     done = true
26.   }
27.
28.   // s'agit-il d'un rechargement de page ?
29.   if (!done && from === to) {
30.     // rien à faire
31.     done = true
32.   }
33.
34.   // contrôle de la navigation du serveur [nuxt]
35.   // on se calque sur le menu de navigation du client
36.
37.   // on traite d'abord le cas de fin-session
38.   if (!done && store.state.started && store.state.authenticated && to === 'fin-session') {
39.     // on nettoie la session
40.     const session = context.app.$session()
```

```
41.       session.reset(context)
42.       // on redirige vers la page index
43.       redirection = 'index'
44.       // travail terminé
45.       done = true
46.     }
47.
48.     // cas où la session PHP n'a pas démarré
49.     if (!done && !store.state.started && to !== 'index') {
50.       // redirection vers [index]
51.       redirection = 'index'
52.       // travail terminé
53.       done = true
54.     }
55.
56.     // cas où l'utilisateur n'est pas authentifié
57.     if (!done && store.state.started && !store.state.authenticated && to !== 'index') {
58.       redirection = 'index'
59.       // travail terminé
60.       done = true
61.     }
62.
63.     // cas où [adminData] n'a pas été obtenu
64.     if (!done && store.state.started && store.state.authenticated && !store.state.métier.taxAdminData && to !== 'index') {
65.       // redirection vers [index]
66.       redirection = 'index'
67.       // travail terminé
68.       done = true
69.     }
70.
71.     // cas où [adminData] a été obtenu
72.     if (
73.       !done &&
74.       store.state.started &&
75.       store.state.authenticated &&
76.       store.state.métier.taxAdminData &&
77.       to !== 'calcul-impot' &&
78.       to !== 'liste-des-simulations'
79.     ) {
80.       // on reste sur la même page
81.       redirection = from
82.       // travail terminé
83.       done = true
84.     }
85.
86.     // on a normalement fait tous les contrôles ---------------------
87.     // redirection ?
88.     if (redirection) {
89.       // on note la redirection dans le store
90.       store.commit('replace', { serverRedirection: true })
91.     } else {
92.       // pas de redirection
93.       store.commit('replace', { serverRedirection: false, from: to })
94.     }
95.     // on sauvegarde le store dans la session [nuxt]
96.     const session = context.app.$session()
97.     session.value.store = store.state
98.     session.save(context)
99.     // on fait l'éventuelle redirection du serveur [nuxt]
100.    if (redirection) {
101.      context.redirect({ name: redirection })
102.    }
103.}
```

- nous reprenons dans ce script les idées déjà développées et utilisées dans le routage du serveur **[nuxt]** de l'application **[nuxt-13]** ;
- nous ajoutons deux propriétés au store de l'application :
  - **[from]** : le nom de la dernière page affichée. On sait que le client **[nuxt]** a cette information mais pas le serveur **[nuxt]**. On va lui ajouter cette information en stockant dans le store, à chaque routage du serveur **[nuxt]**, le nom de la page qui va être affichée. On fera de même à chaque routage du client **[nuxt]**. Ainsi au routage suivant du serveur **[nuxt]**, celui-ci trouvera dans le store, le nom de la dernière page affichée par l'application ;
  - **[serverRedirection]** : lorsqu'une destination de routage sera refusée par le serveur **[nuxt]**, celui-ci opèrera une redirection. Il indiquera alors dans le store que la prochaine destination du serveur **[nuxt]** est une page de redirection. Cette redirection va provoquer une nouvelle exécution du routeur du serveur **[nuxt]**. Si celui-ci découvre que la destination en cours est issue d'une redirection, il laissera faire ;
- lignes 6-11 : on récupère les informations utiles pour le routage ;
- lignes 13-16 : la cible **[/fin-session]** n'est pas associée à une page qui s'appellerait **[fin-session]**. Elle n'a donc pas de nom. On lui en donne un ;
- ligne 18 : la cible d'une éventuelle redirection ;
- ligne 20 : **[done=true]** lorsque les tests du routage sont terminés ;

- lignes 22-26 : comme il a été dit, si le routage en cours est issu d'une redirection, il n'y a rien à faire. En effet, lors du routage précédent, le routeur a décidé qu'il fallait rediriger le navigateur client. Il n'y a pas lieu de reconsidérer cette décision ;
- lignes 28-32 : s'il s'agit d'un rechargement de page, on laisse faire. Ce n'est pas un axiome valable pour toute application **[nuxt]** : il faut regarder pour chaque page les effets d'un rechargement. Ici, il se trouve que le rechargement des pages **[index, calcul-impot, liste-des-simulations]** ne provoque pas d'effets indésirables ;
- lignes 34-84 : le routage du serveur **[nuxt]** reprend le routage du client **[nuxt]**. Lorsqu'on est sur une page, le routage du serveur **[nuxt]** doit refléter le menu de navigation offert par le client **[nuxt]** lorsqu'on est sur cette page ;
- lignes 37-46 : on traite d'abord le cas de la cible **[fin-session]** qui ne correspond pas à une page existante. Si les conditions sont réunies (session démarrée, utilisateur authentifié), on nettoie la session et on redirige l'utilisateur vers la page **[index]** ;
- lignes 48-54 : si la session jSON avec le serveur de calcul de l'impôt n'a pas commencé, alors la seule destination possible est la page **[index]** ;
- lignes 56-61 : si la session jSON a été démarrée et que l'utilisateur n'est pas authentifié et qu'il n'a pas demandé la page d'authentification, alors on redirige vers la page d'authentification qui est la page **[index]** ;
- lignes 63-69 : si l'utilisateur est authentifié mais que la donnée **[adminData]** n'a pas été obtenue, alors on redirige vers la page d'authentification. L'authentification fait deux choses : elle authentifie et si l'authentification a réussi elle demande de plus la donnée **[adminData]**. Si cette dernière n'a pas été obtenue alors il faut recommencer l'authentification ;
- lignes 71-84 : si la donnée **[adminData]** a été obtenue, alors les seules cibles possibles sont **[calcul-impot]** et **[liste-des-simulations]**. Si ce n'est pas le cas, on refuse le routage ;
- lignes 87-94 : on met à jour le store selon qu'il va y avoir redirection ou non ;
- ligne 93 : il n'y a pas redirection. Aussi le **[to]** actuel va devenir le **[from]** du prochain routage ;
- lignes 95-98 : les informations du store sont sauvegardées dans le cookie de la session **[nuxt]** ;
- lignes 99-102 : si on doit faire une redirection, on la fait ;

Pour faire les tests, il faut prendre soin de partir d'une situation vierge en supprimant le cookie de la session **[nuxt]** et le cookie de la session PHP avec le serveur de calcul de l'impôt :



Pour tester le routage du serveur **[nuxt]**, sur chaque page essayez toutes les URL possibles **[/, /calcul-impot, /liste-des-simulations]**. A chaque fois, l'application doit rester dans un état cohérent.

4.17.10 étape 9

L'étape 9 est celle du déploiement de l'application **[nuxt-20]**. Celle-ci nécessite un hébergement offrant un environnement **[node.js]** pour exécuter le serveur **[nuxt]**. Je ne l'ai pas. Le lecteur pourra suivre les procédures décrites au paragraphe lien, pour déployer l'application **[nuxt-20]** sur sa machine de développement et la sécuriser avec un protocole HTTPS.

4.17.11 Conclusion

Le portage de l'application **[vuejs-22]** vers l'application **[nuxt-20]** est désormais terminé. Retenons quelques points de ce portage :

- les pages de **[vuejs-22]** ont été gardées ;
- les opérations asynchrones qui existaient dans les pages de **[vuejs-22]** ont été migrées dans une fonction **[asyncData]** ;
- dans **[nuxt-20]** il a fallu gérer deux entités : le client **[nuxt]** et le serveur **[nuxt]**. Cette dernière entité n'existait pas dans **[vuejs-22]**. Pour garder une cohérence entre les deux entités, nous avons eu besoin d'une session **[nuxt]** ;
- il nous a fallu gérer le routage du serveur **[nuxt]** ;

Dans la pratique, il est sans doute préférable de démarrer directement sur une architecture **[nuxt]** que de faire une architecture **[vue.js]** portée ensuite dans un environnement **[nuxt]**.