

PHP Premiers pas

Support de cours Version 1.3

Sommaire

Sommaire	2
1 Introduction.....	7
1.1 Petite et grande histoire.....	7
1.2 Un langage multi-casquettes.....	9
2 Premiers pas avec PHP	12
2.1 Avec quels outils démarrer ?.....	12
2.2 Choisir un bon IDE	17
2.3 Choisir un stack PHP	21
2.3.1 Le stack ?AMP.....	21
2.3.2 Ligne de commande et serveur web intégré.....	24
2.3.3 Environnements virtualisés	26
2.4 Choisir un outil de test	28
3 Découverte du langage.....	29
3.1 Test du stack PHP	29
3.2 Premier « hello world ».....	33
3.3 Quelques astuces à connaître	41
3.3.1 Astuces d'un haricot du net	41
3.3.2 Index ou pas index ?	42
3.4 Mon « hello world » dans mon smartphone.....	43
3.5 Second « hello world » avec du style	46
3.6 Troisième « hello world » pour découvrir les variables	49
3.7 Quatrième « hello world », avec un peu de JS	51
3.8 Dans les coulisses d'un « hello world »	52
3.9 Déconstruire un « hello world ».....	59
4 Les bases du langage PHP	64
4.1 Les chaînes de caractères	64
4.2 Introduction aux variables, constantes, conditions, boucles	67
4.2.1 Variables, types, conditions.....	67
4.2.2 Constantes.....	75

4.2.3 Tableaux et boucles.....	76
4.2.4 Commentaires	84
4.2.5 Fonctions	85
4.2.6 Include et Require	89
4.3 Compléments sur PHP	90
4.3.1 Dates.....	90
4.3.2 Sessions et cookies.....	96
4.4 Les formulaires et les tableaux \$_GET et \$_POST.....	105
4.4.1 Le tableau \$_GET.....	105
4.4.2 Le tableau \$_POST.....	110
4.4.3 Un formulaire plus intelligent	118
4.4.4 Un formulaire robuste et sécurisé.....	149
4.4.5 Générateur de formulaire	153
4.5 Les objets en PHP	162
4.5.1 Classe « découverte ».....	162
4.5.2 Classe abstraites et méthodes statiques.....	172
4.5.3 Astuces d'héritier	173
4.5.4 Interfaces.....	175
4.5.5 Méthodes magiques.....	176
4.5.6 Trucs et astuces	177
4.5.7 Espaces de nommage	180
4.5.8 SPL (Standard PHP Library).....	183
4.5.9 Design Patterns	185
4.5.10 Quoi d'autre ?	186
4.6 PHP et le SQL	187
4.6.1 Présentation de PDO	187
4.6.2 Connexion PDO et premiers exemples.....	188
4.6.3 Les différents modes Fetch de PDO	191
4.6.4 SQL et sécurité.....	195
4.6.5 Connexion à différents SGBD	196

4.6.6 What else ?	198
4.7 Composer	199
4.8 Les fichiers	200
4.8.1 Les fichiers texte.....	200
4.8.2 Les fichiers CSV	203
4.8.3 Les fichiers XML.....	205
4.8.4 Les fichiers PDF.....	210
4.9 Etudes de cas.....	211
4.9.1 Préparation d'un projet.....	211
4.9.2 Liste avec pagination	218
4.9.3 Sous-requêtes scalaires, données temporelles.....	235
4.9.4 Mini-CRUD, mais il fait le maximum.....	253
4.9.5 Pistes pour étudier l'architecture MVC.....	265
4.9.6 Manuel du petit Scraper.....	266
5 Annexe.....	274
5.1 Opérateurs.....	274
5.2 Préparation d'un jeu de données SQL avec Excel	277
5.3 Tables des pays au format SQL.....	279
5.4 Mesure de performances	285
5.5 Les filtres	286
5.6 Les pièges du « select multiple ».....	289
5.7 Une classe DBWrapper pour MySQL.....	294
5.8 Jeux de données du chapitre 4.9.3.....	298
5.9 PHP Manifesto.....	303
6. Bibliographie et Liens utiles.....	304
7 Changelog	306

Notes de l'auteur :

Je m'appelle Grégory Jarrige.

Je suis développeur professionnel depuis 1991. Après avoir longtemps travaillé sur des gros systèmes et des langages et technos propriétaires, j'ai fait le pari de me former aux technos et langage open source vers 2005-2006. J'ai commencé à développer des applications webs professionnelles à partir de 2007, avant d'en faire mon activité principale à partir de 2010. L'arrivée du HTML5 dans la même période a été pour moi une véritable bénédiction, et surtout un formidable terrain d'expérimentation (avec des API comme Canvas, WebAudio, etc...).

En plus de mon activité de développeur freelance, je suis également formateur - sur des sujets tels que PHP, HTML5, Javascript, SQL – tantôt en entreprise, tantôt dans le cadre de programmes de reconversion (GRETA notamment).

Le présent document est publié sous Licence Creative Commons n° 6.

Il est disponible en téléchargement libre sur mon compte Github :

<https://github.com/gregja/PHPCorner>

Ce cours se situe dans la continuité des supports « HTML5 – Premiers pas », « CSS3 – Premiers pas » et « Javascript – Premiers pas » qui sont disponibles dans le dépôt Github suivant :

<https://github.com/gregja/JSCorner>

Ce cours peut servir de préparation (ou de remise à niveau), pour ceux qui souhaiteraient étudier par la suite mon cours sur le micro-framework SILEX :

https://github.com/gregja/Cours_SILEX_4_Biz

« Je peux tout te montrer mais je ne peux rien t'apprendre. »

Extrait du très beau film documentaire « Être et devenir », de Clara Bellar.

J'ai découvert ce documentaire au moment même où je commençais à rédiger ce support de cours. Cette phrase – d'un maître luthier à son apprenti - m'a impressionné, car elle répondait en écho à mes propres préoccupations d'enseignant.

Comment transmettre sans noyer l'apprenant sous un déluge d'informations qu'il n'est pas en mesure de digérer ? Non pas que l'apprenant soit idiot, mais même quand l'enseignant s'efforce de développer une progression pédagogique, deux fois sur trois les informations arrivent dans un ordre qui ne correspond pas à ce que l'apprenant est en mesure d'assimiler, et de comprendre. En termes de niveau de connaissance, tout le monde ne part pas du même point et n'avance pas au même rythme, chaque apprenant est unique. Plus grave encore, les informations trop « prémâchées » ne rendent pas service à l'apprenant. Trop souvent, l'apprenant est mis devant le fait accompli, ne comprend pas pourquoi l'enseignant arrive à certaines conclusions, ou a pris certains raccourcis, et il décroche.

A quel moment est-ce que j'ai le plus progressé dans ma maîtrise de la programmation ? En général, c'est en croisant la route d'excellents développeurs qui écrivaient des choses épatales. Ils ne m'expliquaient rien, mais leur code était suffisamment clair et démonstratif pour que je sache quelle était la voie que je devais suivre.

Alors, « tout montrer » ? Vraiment ? Un support de cours comme celui-ci a ses limites, la plus importante étant la limite de temps (temps d'écriture pour moi, et de lecture pour l'apprenant). Alors j'ai dû faire des choix, en me concentrant en priorité sur des points qui sont trop rarement expliqués dans les nombreux livres et tutos que j'ai lus par le passé (même si beaucoup étaient de très bons supports qui m'ont beaucoup apporté). Ce support de cours, c'est un peu celui que j'aurais aimé avoir si j'avais été débutant aujourd'hui.

1 Introduction

1.1 Petite et grande histoire

PHP est un vieux langage, si l'on considère qu'il est apparu au milieu des années 90, à peu près en même temps que Ruby et Javascript.

PHP a été développé par Rasmus Lerdorf, qui souhaitait développer un langage facilitant la gestion de pages HTML. Dans un premier temps, Rasmus a utilisé PHP pour gérer un compteur de visite sur les pages de son site. Mais très rapidement, Rasmus a étendu les possibilités du langage pour en faire un véritable couteau suisse, particulièrement bien adapté à la production de code HTML. Comme le HTML, ce n'est pas jamais que du texte, il s'avère à l'usage que PHP est une formidable boîte à outils pour la manipulation et la production de tout type de fichier texte (ce que nous étudierons aussi dans le cadre de ce cours).

Ce court historique emprunté à Wikipédia est intéressant :

Rasmus décida alors en 1995 de publier son code, pour que tout le monde puisse l'utiliser et en profiter. PHP s'appelait alors PHP/FI (pour Personal Home Page Tools/Form Interpreter). En 1997, deux étudiants, Andi Gutmans et Zeev Suraski, redéveloppèrent le cœur de PHP/FI. Ce travail aboutit un an plus tard à la version 3 de PHP, devenu alors PHP: Hypertext Preprocessor. Peu de temps après, Andi Gutmans et Zeev Suraski commencèrent la réécriture du moteur interne de PHP. Ce fut ce nouveau moteur, appelé Zend Engine — le mot Zend est la contraction de Zeev et Andi — qui servit de base à la version 4 de PHP.

Source : <https://fr.wikipedia.org/wiki/PHP>

On peut compléter cette précision en disant que, quelques années plus tard, Zend réécrivit de nouveau le moteur de PHP, de manière à implémenter un modèle objet plus complet que celui proposé par PHP4, et plus proche du modèle objet de Java. Cette réécriture aboutit à la publication de PHP5, en 2004. La version 5.3 est généralement considérée comme la version qui a fait entrer PHP dans le mode du développement professionnel, avec de nombreuses nouveautés intéressantes (tels que les espaces de nommage, les fonctions anonymes, etc...).

Parmi les points qui ont contribué au succès du langage :

- sa gratuité

- sa simplicité de mise en œuvre, avec un mode de fonctionnement s'appuyant sur le serveur d'application Apache, et simplifiant toute la mécanique d'échange entre serveur et navigateur (via le protocole HTTP) : PHP est en effet configuré pour produire du HTML par défaut, sauf si on lui demande de produire autre chose (XML, texte, etc...).
- La facilité avec laquelle on peut combiner code HTML et code PHP au sein d'un même code source (comme on le verra dans quelques exemples), a aussi contribué à populariser PHP,
- La disponibilité d'un connecteur gratuit avec la base de données MySQL (elle aussi gratuite), suivi très rapidement de connecteurs gratuits pour la plupart des autres bases de données du marché

Les hébergeurs ont très tôt compris le potentiel de PHP, et se sont livrés à une concurrence féroce en proposant des serveurs mutualisés préconfigurés avec Apache, PHP et MySQL. Et cela pour des tarifs très avantageux, pour le client final. On trouve dans ce domaine des acteurs bien établis, comme OVH, Gandi, 1&1, etc... et on voit encore régulièrement de nouveaux challengers apparaître. Mais ceci est une autre histoire...

Le langage PHP a continué à évoluer lentement. On notera une tentative avortée d'écrire un PHP6 : il s'agissait de réécrire l'interpréteur de PHP de manière à ce qu'il supporte nativement la norme d'encodage UTF-8. Mais ce projet était trop ambitieux, et la communauté des développeurs du langage PHP s'est épuisée pendant une période sur ce projet, avant de l'abandonner.

Vers la fin des années 2000, la société Facebook - qui utilisait massivement le langage PHP pour son application phare (réseau social) - décida d'écrire son propre interpréteur PHP, afin d'obtenir de meilleures performances que celles obtenues avec l'interpréteur standard développé par Zend. Cela aboutit à deux projets distincts mais étroitement liés :

- le projet HHVM (Hip Hop Virtual Machine) : <http://hhvm.com/>
- le projet Hack, qui est un langage PHP à la sauce Facebook : <http://hacklang.org/>

Facebook est même allée – pour le développement de Hack – jusqu'à écrire une spécification du langage PHP, ce que Zend n'avait jamais pris la peine de faire :

<https://github.com/php/php-langspec>

Facebook publiant régulièrement des benchmarks dans lesquels elle affirmait obtenir de meilleures performances que l'interpréteur PHP standard, la société Zend se vit obligée de réagir, en publiant en 2014 le projet PHPNG (NG pour « New Generation »). Il semble que les projets

HHVM (qui continue à être utilisé par Facebook) et PHPNG (qui deviendra par la suite PHP7) soient relativement proches en termes de performance.

On notera que Zend a été rachetée en 2015 par la société Rogue Wave Software, qui a repris le catalogue de solutions proposées par Zend et continue à les faire évoluer.

Le langage PHP continue à évoluer, sous la houlette d'un groupe d'experts du langage, et on peut suivre cette évolution en consultant la changelog de PHP7 :

<http://php.net/ChangeLog-7.php>

1.2 Un langage multi-casquettes

Le langage PHP a connu de nombreux changements, mais une constante dans cette évolution, c'est que ses concepteurs se sont efforcés de maintenir une compatibilité ascendante entre les différentes versions. Par exemple, l'arrivée avec PHP5 d'un modèle objet plus avancé que le modèle de PHP4, s'est fait en douceur, les 2 modèles ayant pu cohabiter assez longtemps pour que les développeurs aient le temps d'adapter leurs applications.

Un exemple de classe en style PHP4 :

```
class toto {
    var $machin = '';
    // méthode constructeur en PHP4
    function toto () {
    }
    // méthode quelconque
    function getMachin() {
        return $this->machin;
    }
}
```

La même classe réécrite en PHP5 :

```
class toto {
    public $machin = '';
    // méthode constructeur en PHP5
    public function __construct() {
    }
    // méthode quelconque
    public function getMachin() {
        return $this->machin;
    }
}
```

Cette compatibilité ascendante a aussi un certain prix, à savoir le manque de cohérence entre des fonctions apparentées. Comment expliquer autrement la différence dans l'ordre des paramètres, entre des fonctions comme `str_replace()` et `strr()`? Et pourquoi certaines fonctions de manipulation de chaînes de caractères ont un nom qui commence par « `str_` » et d'autres pas ? On retrouve d'ailleurs le même manque de cohérence dans les fonctions de manipulation de tableaux.

On a pu constater au fil du temps une certaine inertie de la part de la communauté des développeurs, certains d'entre eux tardant à prendre en compte les recommandations des concepteurs du langage.

Par exemple, il semble incroyable que l'on trouve encore aujourd'hui sur internet des tutoriels de ce type :

```
$connect = mysql_connect('localhost','root','');
    or die ("erreur de connexion");
mysql_select_db('base',$connect) or die ("erreur de connexion base");
```

L'annonce du retrait de l'extension « `mysql` » avait été annoncée de longue date, elle a été officialisée avec PHP 5.5, et son retrait est devenu effectif avec PHP7. Les raisons du retrait de cette extension ? Essentiellement des problèmes de performance et des problèmes de sécurité. Les concepteurs de PHP avaient pourtant bien fait les choses, avec les extensions « `mysqli` » et « `pdo_mysql` », toutes deux très bien, et toutes deux disponible depuis longtemps. Bon, on va dire que c'est un accident, ça se passera mieux la prochaine fois 😊. En attendant, si vous tombez sur un tuto qui vous parle de la fonction `mysql_connect()`, passez votre chemin.

L'un des points forts de PHP, c'est sa grande souplesse d'utilisation. On peut en effet écrire son code dans un style procédural, un style objet, ou une combinaison des deux.

On peut utiliser PHP pour :

- le développement de sites webs,
- la production et la consommation de services webs (via le protocole SOAP ou l'architecture REST),
- le développement de traitements batchs,
- la conversion de fichiers (par exemple de CSV vers XML, ou l'inverse),
- de la reprise de données (par exemple de fichiers XML ou CSV vers une base de données MySQL ou PostgreSQL, ou d'une base DB2 vers une base PostgreSQL, etc...)
- le réencodage de données (via des fonctions de conversion spécifiques),
- la génération de code (ça peut être du code PHP, du code SQL, ou autre...)
- etc...

Bref, PHP est un formidable « couteau suisse », qui peut vous rendre service quotidiennement pour un grand nombre d'usages différents.

Ce support de cours est une introduction à PHP, il n'était pas possible de couvrir tous les sujets (à moins de vouloir écrire un ouvrage de 1000 pages, ce qui n'est pas l'objectif). Des conseils de lecture vous seront fournis au fil de l'eau, pour vous permettre d'approfondir certains sujets par d'autres voies.

2 Premiers pas avec PHP

2.1 Avec quels outils démarrer ?

Nous avons vu dans les cours HTML5 et Javascript que l'on peut utiliser des solutions en ligne très pratiques pour développer et prototyper du code HTML5 et JS, telles que les plateformes Codepen et Codecircle.

- Codepen : <https://codepen.io/>
- Codecircle : <https://live.codecircle.com/>

Malheureusement, il n'existe pas de solutions aussi souples pour PHP, et c'est bien dommage. On peut cependant utiliser le site internet « PHP Sandbox » (traduisez : « bac à sable PHP ») :

<http://sandbox.onlinephpfunctions.com/>

The screenshot shows the homepage of Online PHP Functions. At the top, there's a navigation bar with links for Home, Sandbox, PHP Functions (with a dropdown menu), Custom Functions, Tutorials, PHP Benchmarks, and About. Below the navigation, a banner reads "Online PHP function(s){ #Test PHP Functions online}".

The main content area is titled "PHP Sandbox" and contains the sub-instruction "Test your PHP code with this code tester". A note below says "You can test your PHP code here on many php versions." Underneath, there's a section labeled "Your script:" containing a code editor with the following PHP code:

```

1  <?php
2      //Enter your code here, enjoy!
3
4  $array = array("1" => "PHP code tester Sandbox Online",
5                 "foo" => "bar", 5 , 5 => 89009,
6                 "case" => "Random Stuff: " . rand(100,999),
7                 "PHP Version" => phpversion()
8                 );
9
10
11
12
13
14

```

Très pratique, ce « bac à sable » permet de saisir et tester du code PHP en ligne. Cela ne permet pas de développer des applications, mais pour tester quelques fonctions ou de petits bouts de code « maison », c'est vraiment sympa.

A signaler toutefois l'existence d'une plateforme dédiée à l'apprentissage de PHP, dont je découvre l'existence au moment même où je rédige ces pages :

<http://codeclassroom.net/>

... mais il semble que la plateforme ne soit pas encore opérationnelle, à suivre donc...



Tools to learn to code. For schools and beginners.

LEARN & CODE AT THE SAME TIME

DOWNLOAD for WINDOWS
version 14.1 alpha - soon



STUDENTS

Enter the library
and learn something

TEACHERS

Open your office, create courses
and enrich the library

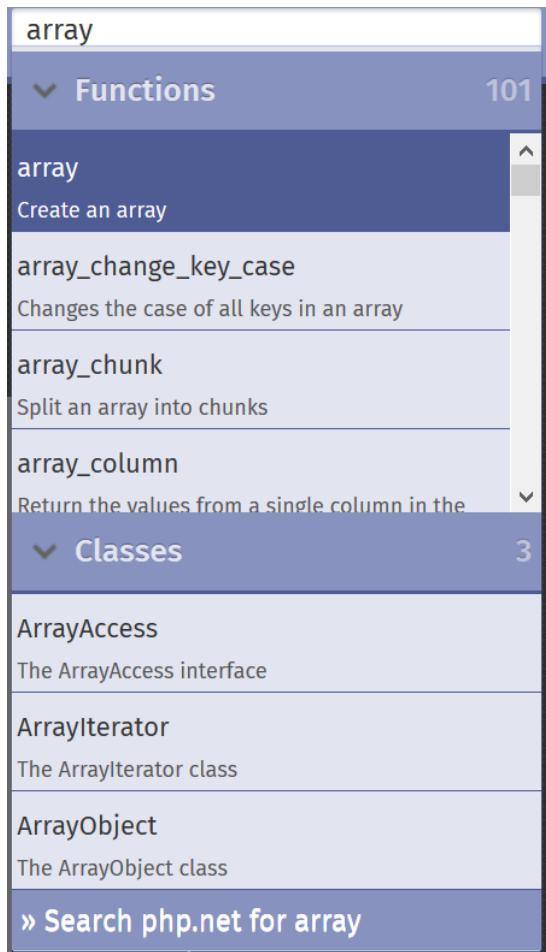
```
1 <!DOCTYPE html>
2 <html>
3   <head>
4   </head>
5   <body>
6     <p>
7       <?php
8         echo "Hello";
9       ?>
10      </p>
11    </body>
12  </html>
```

A complete and fully interactive environment to help students (kids or adults) to learn code and teachers to create and share courses.

PHP HTML CSS JS SQL ...

Dès que nous allons commencer à développer, nous allons avoir besoin de documentation.
La documentation officielle de PHP est un incontournable :
<http://php.net/>

La fonction de recherche du site permet de retrouver tout ce qu'on veut très rapidement :



La documentation du site php.net est très complète, et le site est traduit en plusieurs langues.

Dans l'exemple ci-dessous, on est en train de consulter la documentation de la fonction PHP « `array_column` », que l'on peut faire défiler et dans laquelle on trouvera un descriptif détaillé suivi de nombreux exemples.

Dans la partie de droite (sur fond noir), on peut faire défiler la liste de toutes les fonctions dont le nom commence par « `array` » (fonctions dédiées à la manipulation de tableaux), et vous verrez qu'il y en a une tétrach... enfin bref, un gros paquet, quoi ☺.

The screenshot shows a screenshot of the PHP documentation website. At the top, there's a navigation bar with links for 'Downloads', 'Documentation' (which is currently selected), 'Get Involved', and 'Help'. A search bar is also at the top right. Below the navigation, a breadcrumb trail shows the path: 'Manuel PHP' > 'Référence des fonctions' > 'Extensions relatives aux variables et aux types' > 'Les tableaux' > 'Fonctions sur les tableaux'. The main content area is titled 'array_column'. It includes a 'Description' section with the PHP code signature: `array array_column (array $array , mixed $column_key [, mixed $index_key = null])`. To the right of the main content, there's a sidebar titled 'Fonctions sur les tableaux' which lists various array-related functions like 'array_change_key_case', 'array_chunk', etc. A language selection dropdown is also visible on the right side of the main content area.

Autre site de référence intéressant pour l'étude de PHP, W3Schools, qui consacre une partie de son site à ce langage :

<https://www.w3schools.com/php/default.asp>

En termes de logiciels, nous allons avoir besoin de 2 choses importantes, et même d'une troisième :

- un bon éditeur de code, ou IDE (pour Integrated Development Environment)
- un stack PHP comprenant Apache, PHP et MySQL
- un (ou plusieurs) outil(s) de test

Nous allons préciser ces 3 points dans les chapitres suivants.

Mais j'aimerais vous montrer que vous pouvez faire énormément de choses à l'intérieur même de votre navigateur préféré, sans utiliser le moindre outil extérieur.

Nous aurons cependant besoin, dans la suite de ce cours, d'une page HTML de base, la plus simple possible, histoire de démarrer notre apprentissage sur de bonnes bases. Le plus simple, pour obtenir cette page HTML, c'est d'en créer une. Voici son code source, vous pouvez le copier-coller dans un éditeur quelconque, en prenant soin de le sauvegarder avec l'extension « .html ». Vous pouvez par exemple l'appeler « mapage.html » :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>ma page de test</title>
</head>
<body>
    <div id="cible">ceci est le contenu d'une balise "div" de test</div>
</body>
</html>
```

Si vous ne comprenez pas certains éléments du code ci-dessus, je vous recommande d'étudier le cours « HTML5 – Premiers pas », qui est disponible dans le même projet Github.

2.2 Choisir un bon IDE

En termes d'éditeurs et d'environnement de développement, on dispose d'un panel de solutions assez large.

L'éditeur Notepad++ est un bon éditeur de code, très polyvalent, qui propose une coloration syntaxique immédiate, dès qu'il détecte que le fichier ouvert est un source HTML ou PHP (il le détermine grâce à l'extension du fichier) :

<https://notepad-plus-plus.org/fr/>

Pour nos premiers tests en PHP, nous utiliserons Notepad++. Il est agréable à utiliser, léger, peu gourmand en ressources, bref, c'est bon pour la planète 😊.

SublimeText est un autre éditeur de code très puissant et très polyvalent, qui est tout à fait adapté à la maintenance de code PHP, à condition de lui ajouter un certain nombre de plugins :

<https://www.sublimetext.com/>

On trouve sur le web, et sur youtube, des tutos indiquant quels plugins installer pour travailler avec PHP.

Netbeans est un IDE gratuit (open source) proposé par Oracle. Très complet, il est parfaitement adapté à la maintenance de code PHP, Javascript et HTML5 :

<https://netbeans.org/>

On peut télécharger une version personnalisée en fonction du type de code que l'on souhaite développer (HTML5 et PHP uniquement ? Java ?) :

The screenshot shows the NetBeans IDE 8.2 Download page. At the top, there are fields for Email address (optional), IDE Language (English), Platform (Windows), and checkboxes for newsletter subscription and contact information. Below this is a note about unsupported technologies for Windows.

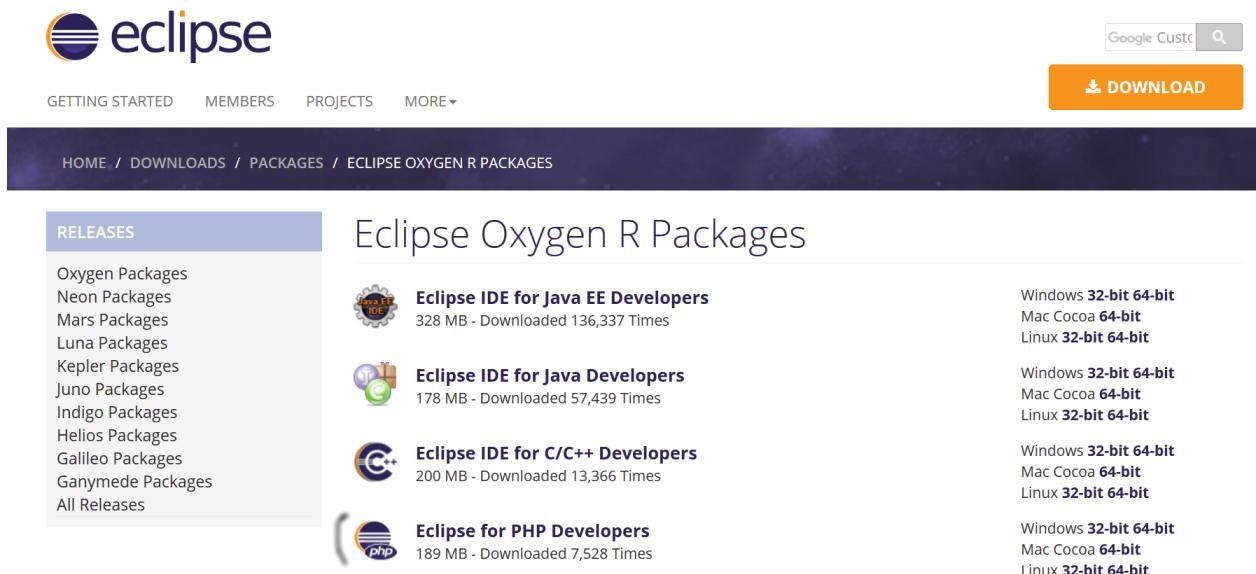
The main section is titled "NetBeans IDE Download Bundles" and contains a matrix table:

Supported technologies *	Java SE	Java EE	HTML5/Javascript	PHP	C/C++	All
NetBeans Platform SDK	•	•				•
Java SE	•	•				•
Java FX	•	•				•
Java EE		•				•
Java ME						•
HTML5/Javascript		•	•	•		•
PHP			•	•		•
C/C++					•	•
Groovy						•
Java Card™ 3 Connected Bundled servers						•
GlassFish Server Open Source Edition 4.1.1		•				•
Apache Tomcat 8.0.27		•				•

Below the table are several "Download" buttons for different platforms and file types (x86, x64, etc.). At the bottom, there is a note about file sizes: Free, 95 MB; Free, 197 MB; Free, 108 - 112 MB; Free, 108 - 112 MB; Free, 107 - 110 MB; Free, 221 MB.

La version HTML5 et PHP consomme moins de mémoire, qu'une version plus complète, ça aussi c'est bon pour la planète.

Eclipse est aussi un IDE très complet, gratuit, qui se décline en différentes versions :
<http://www.eclipse.org/downloads/packages/>



The screenshot shows the Eclipse website's package download section. At the top, there's a navigation bar with links for 'GETTING STARTED', 'MEMBERS', 'PROJECTS', 'MORE...', 'Google Custom Search', and a large orange 'DOWNLOAD' button. Below the navigation is a breadcrumb trail: 'HOME / DOWNLOADS / PACKAGES / ECLIPSE OXYGEN R PACKAGES'. The main content area has a title 'Eclipse Oxygen R Packages'. On the left, a sidebar titled 'RELEASES' lists various Eclipse releases: Oxygen Packages, Neon Packages, Mars Packages, Luna Packages, Kepler Packages, Juno Packages, Indigo Packages, Helios Packages, Galileo Packages, Ganymede Packages, and All Releases. The main content area displays four packages: 'Eclipse IDE for Java EE Developers' (328 MB, 136,337 downloads), 'Eclipse IDE for Java Developers' (178 MB, 57,439 downloads), 'Eclipse IDE for C/C++ Developers' (200 MB, 13,366 downloads), and 'Eclipse for PHP Developers' (189 MB, 7,528 downloads). Each package entry includes a small icon, the package name, its size, and the number of downloads.

Enfin, PHPStorm est un excellent IDE développé par la société Jetbrains. Très complet, il est très apprécié des développeurs. Malheureusement il n'est pas gratuit, et le coût de sa licence annuelle est peut être un peu élevé, si l'on est développeur amateur, ou étudiant :
<https://www.jetbrains.com/phpstorm/>

Pour ceux qui décideraient d'utiliser Netbeans, voici quelques conseils pour améliorer l'ergonomie de l'IDE :

Allez dans le menu "Tools -> Options"
... puis sélectionnez l'onglet "Appearance"

On peut cocher les options suivantes :

- same background color for files from the same project
- show parent folder name in tab title (pas obligé mais intéressant)
- show full file path

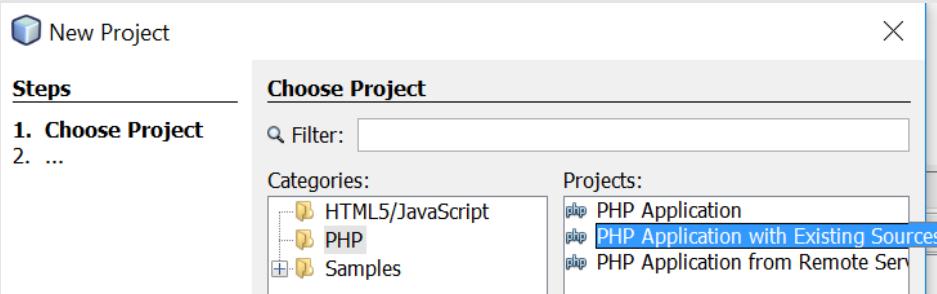
Toutes ces options sont facultatives, mais elles peuvent améliorer votre confort d'utilisation, aussi je vous invite à les tester, et à retenir celles qui vous conviennent le mieux.

On peut aussi cocher l'option "Multi-row tabs" et sélectionner au choix :

- maximum row count (mettre une valeur de 2 ou 3 maximum)
- one row per project (que je trouve plus pratique)

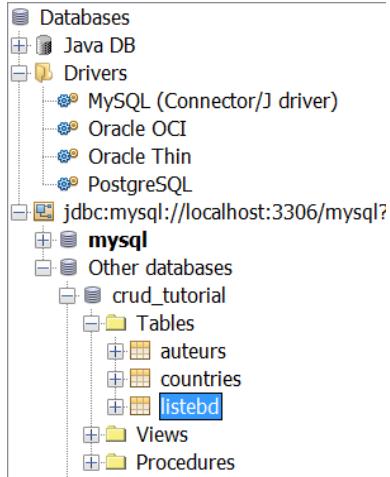
Dans le menu "Fonts & colors" on peut changer le look général en choisissant un profil différent (par exemple "Northway Today")

On reconnaît qu'un projet a déjà été ouvert dans Netbeans au fait qu'il contient un répertoire « nbproject ». Il s'agit d'un répertoire contenant des fichiers de configuration générés par Netbeans lui-même. Dans ce cas de figure, on peut ouvrir le projet dans Netbeans en passant par l'option de menu « File -> Open Project ». Si le projet ne contient pas encore de répertoire « nbproject », alors il faut passer par l'option de menu « File -> New Project » et choisir l'option « PHP Application with Existing Sources »



... sauf si on souhaite créer un projet de toutes pièces en passant par Netbeans, auquel cas on sélectionnera juste au dessus qui s'intitule « PHP Application ».

On notera que la plupart des IDE fournissent des outils permettant de travailler directement sur les bases de données SQL. Par exemple, dans le cas de Netbeans, on dispose en standards de connecteurs pour plusieurs SGBD (Systèmes de Gestion de Bases de Données), comme MySQL, PostgreSQL, Oracle (2 versions), et on peut en ajouter d'autres en cas de besoin :



Dans mon cas, j'ai ouvert une connexion sur mes bases MySQL, je peux donc consulter le contenu des tables et exécuter tout type de requête SQL :

The screenshot shows the NetBeans SQL editor. At the top, the connection is set to 'jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=convertToNull [root on Default schema]'. Below the connection, there are two tabs: the first tab contains the SQL query 'SELECT * FROM crud_tutorial.listebd LIMIT 100;', and the second tab shows the results of this query. The results table has columns '#', 'id', 'titre', 'auteur', and 'editeur'. The data is as follows:

#	id	titre	auteur	editeur
1	1	Garulfo	Ayroles, Maiorana, Leprévost	Delcourt
2	2	Horologiom	Fabrice Lebeault	Delcourt
3	3	Le château des étoiles	Alex Alice	Rue De Sevres
4	4	Le voyage extraordinaire	Camboni, Filippi	Vents d'Ouest

2.3 Choisir un stack PHP

2.3.1 Le stack ?AMP

Il existe de nombreux stacks PHP.

Ce terme de « stack » peut surprendre. Il signifie « pile » en anglais, et un environnement PHP, ce n'est jamais qu'une pile de logiciels, tels que : Apache, MySQL et PHP. L'ensemble de ces 3 logiciels est généralement désigné par le sigle « AMP ».

Si l'on rajoute devant ce sigle le « W » (pour « Windows »), ou le « L » (pour « Linux »), ou le « M » (pour « Mac »), vous comprenez maintenant pourquoi on parle souvent de WAMP, LAMP ou MAMP pour désigner un stack PHP.

Bon, mais c'est quoi au juste, un stack « AMP » :

- Apache, c'est le serveur d'application. Dans le contexte de notre stack PHP, Apache est une sorte de tour de contrôle, qui intercepte les requêtes HTTP provenant de l'internet (ou d'un réseau informatique interne à une entreprise, car tout ne passe pas forcément par internet). Apache fait office de routeur, il reçoit des requêtes HTTP lui demandant de transmettre une information à une application (par exemple de transmettre à un script PHP les données provenant d'un formulaire HTML). Apache s'exécute, et dès que la réponse produite par l'interpréteur PHP est disponible, Apache se charge de retransmettre cette réponse au client (généralement un navigateur internet, mais cela peut aussi être un autre type de logiciel).
- PHP, c'est l'interpréteur PHP : quand Apache reçoit une requête lui demandant d'appeler un script PHP, il vérifie si cette ressource existe, puis fait appel à l'interpréteur PHP. En très résumé, l'interpréteur PHP est un logiciel qui transforme le code source PHP en un code compréhensible par l'ordinateur. Une fois que le script PHP a terminé son travail, l'interpréteur PHP transmet à Apache la réponse produite par ce script. Le plus souvent, la réponse est au format HTML, mais cela peut aussi être un autre format, comme du XML, ou du JSON (nous étudierons ces 2 formats par la suite).
- MySQL, c'est la base de données. Comme son nom l'indique, c'est une base de données qui s'appuie sur un vieux langage d'interrogation, le SQL. Ce vieux langage est toujours beaucoup utilisé pour le stockage de données, tant il est pratique et puissant. Certains développeurs considèrent que le SQL est obsolète, et qu'il n'y a point de salut en dehors des bases de données de type NoSQL (comme MongoDB et quelques autres). C'est un débat stérile, les bases SQL et NoSQL ont toutes deux des avantages et des

inconvénients qu'il serait trop long de développer ici. En ce qui nous concerne, nous travaillerons beaucoup avec MySQL, mais nous jetterons aussi un coup d'œil à PostgreSQL et à MongoDB.

La réalité de ce sigle « AMP » est aujourd’hui battue en brêche, par le fait que ce stack peut être complètement remanié en fonction des besoins et des contraintes d'une entreprise (ou d'un projet) :

- Apache est de plus en plus souvent remplacé par NGinx, projet concurrent souvent présenté comme plus performant
- MySQL est de plus en plus souvent remplacé par MariaDB, qui est un clone open source de MySQL (rappelons simplement que MySQL appartient aujourd’hui à Oracle). MySQL et MariaDB ont le même premier caractère, mais on peut leur préférer d'autres bases de données, SQL ou NoSQL. Parmi les bases de données SQL concurrentes, on trouve PostgreSQL (open source), SQLite (open source), SQL Server (Microsoft), DB2 (IBM), Oracle (Oracle), etc...
- Bon, le seul élément stable dans tout ça, c'est PHP (encore qu'il pourrait être remplacé par Hack, mais seul Facebook fait réellement ça).

Bon, dans un contexte d'apprentissage, un bon vieux stack « AMP » est très satisfaisant.

S'il est possible de télécharger chacun des éléments du stack sur les sites officiels respectifs des projets Apache, MySQL et PHP, et de les installer soi-même manuellement, on peut s'affranchir de cette tâche laborieuse en utilisant un stack prêt à l'emploi... et il en existe plusieurs.

Dans le contexte de Windows, on trouve notamment les stacks suivants :

- Wampserver : <http://www.wampserver.com/>
- Xampp : <https://www.apachefriends.org/fr>
- EasyPHP : <http://www.easyphp.org/>
- UWamp : <https://www.uwamp.com/fr/>

Pour ma part, j'aime bien utiliser Xampp, je le trouve très pratique, et je ne suis pas le seul. Xampp existe pour Windows, pour Mac (OS X) et pour Linux.

Sur Mac, on trouve aussi le projet Mamp, qui remplit le même rôle que Xampp :
<https://www.mamp.info/>

Concernant UWamp : le « U » vient de « USB », car ce stack léger est conçu pour fonctionner à partir d'une simple clé USB, ce qui peut être pratique si on est amené à travailler sur plusieurs machines (ou si on est embêté par des droits administrateur trop restrictifs).

A noter que le projet EasyPHP se décline en 2 projets distincts : EasyPHP WebServer et EasyPHP DevServer.

A noter aussi, le stack WAPP, proposé par la société Bitnami, dans lequel MySQL a été remplacé par PostgreSQL :

<https://bitnami.com/stack/wapp>

La société Zend, qui est aujourd’hui une « Rogue Wave Company » propose son propre stack PHP, le Zend Server :

http://www.zend.com/en/products/zend_server#editions

Zend Server est un stack PHP de niveau professionnel, contenant un ensemble de composants de haut niveau (des outils de déploiement, des outils pour la gestion de travaux asynchrones, des outils pour l’analyse de performance, etc...). Malheureusement, la version « Community Edition » - qui a été proposée pendant une période, et qui était gratuite - n’existe plus, et les licences proposées sont trop chères pour un développeur débutant, étudiant ou pas.

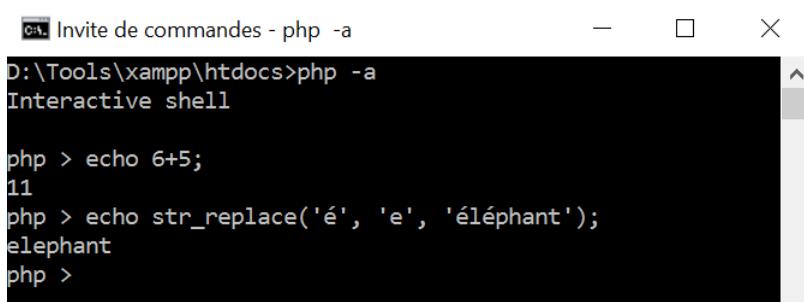
Attention : il ne faut pas confondre « Zend Server » qui est le stack PHP que je viens d’évoquer, avec Zend Framework, qui est un framework de développement concurrent de frameworks comme Symfony, Laravel, Drupal, CakePHP et quelques autres. Beaucoup de personnes font encore aujourd’hui l’amalgame entre ces produits de la marque Zend.

Ce support de cours n’est pas un comparatif sur les stacks PHP, et n’a pas vocation à le devenir 😊. Mais je crois utile de vous sensibiliser sur le fait qu’il existe différents solutions pour faire tourner du code PHP. Dans les chapitres suivants, je vais évoquer brièvement quelques solutions complémentaires à celles que je viens d’évoquer, solutions dont certaines sont apparues assez récemment.

2.3.2 Ligne de commande et serveur web intégré

A partir de la version 5.1, PHP s'est vu doter d'une ligne de commande activable avec la commande :

```
php -a
```



```
D:\Tools\xampp\htdocs>php -a
Interactive shell

php > echo 6+5;
11
php > echo str_replace('é', 'e', 'éléphant');
elephant
php >
```

Cette ligne de commande est pratique, d'autant qu'elle offre diverses options.

Pour en savoir plus :

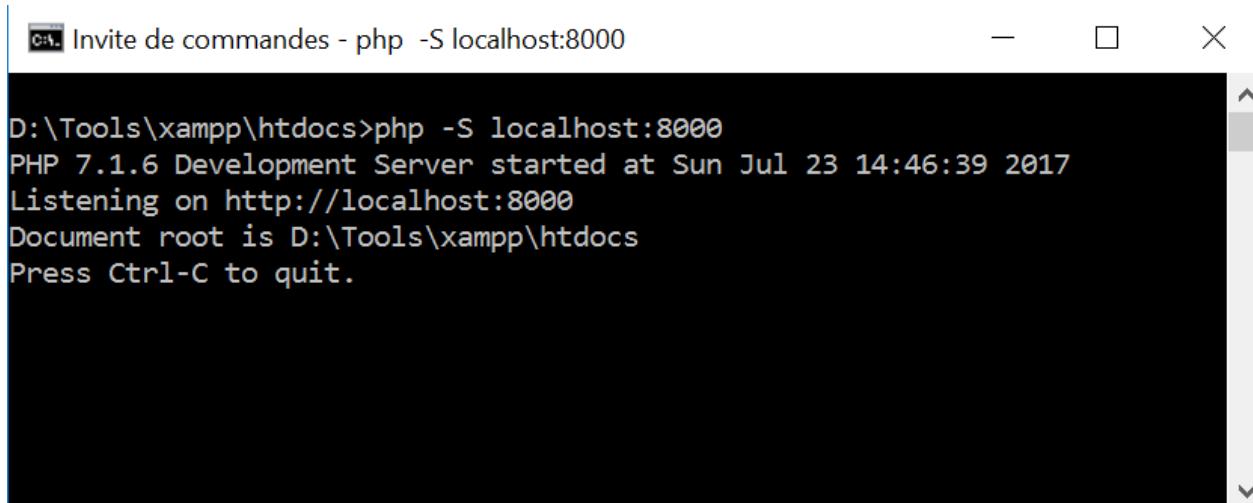
<http://php.net/manual/fr/features.commandline.usage.php>

<https://openclassrooms.com/courses/exécuter-php-en-cli-console>

A partir de la version 5.4, l'interpréteur PHP s'est vu dotter d'un serveur web intégré que l'on peut lancer avec une simple commande. Pour l'illustrer, je vous propose de tenter une petite expérience.

Lancer l'invite de commande Windows, puis positionnez-vous dans le répertoire « htdocs » de votre stack Xampp. Saisissez la ligne ci-dessous :

```
php -S localhost:8000
```

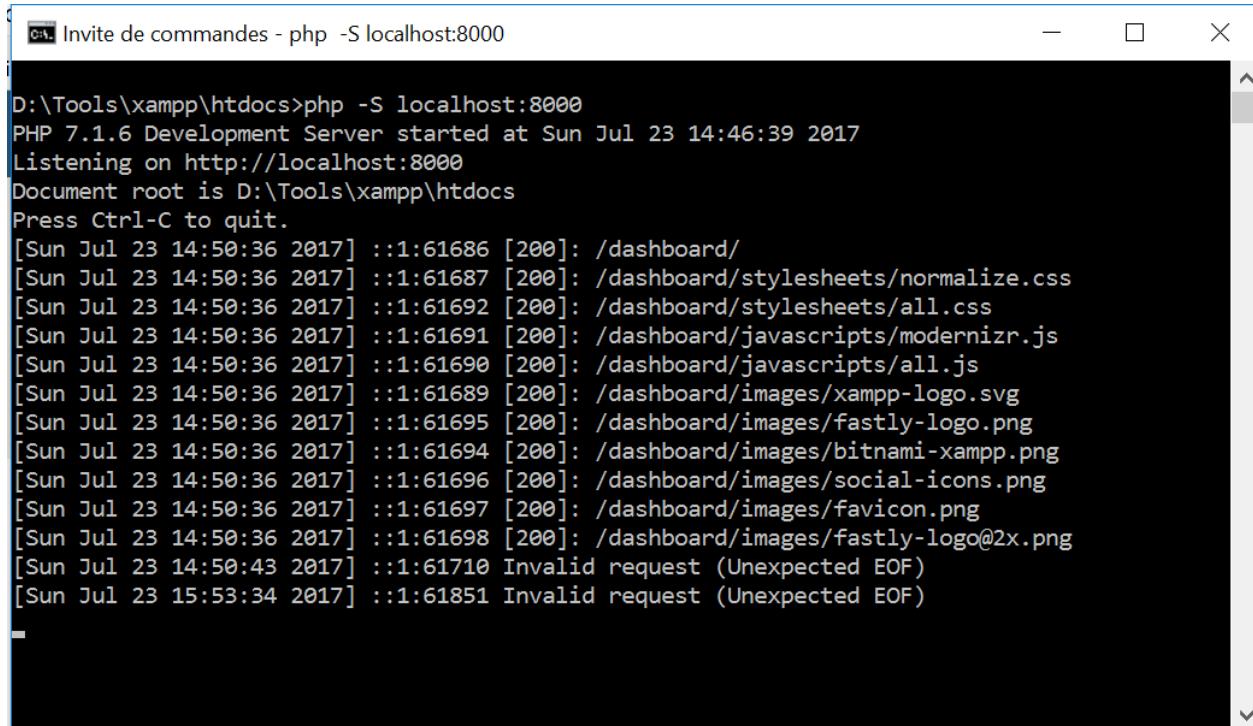


```
D:\Tools\xampp\htdocs>php -S localhost:8000
PHP 7.1.6 Development Server started at Sun Jul 23 14:46:39 2017
Listening on http://localhost:8000
Document root is D:\Tools\xampp\htdocs
Press Ctrl-C to quit.
```

Allez maintenant sur votre navigateur préféré et saisissez ceci :

<http://localhost:8000/>

Vous allez voir s'afficher le « dashboard » (tableau de bord) web de Xampp, mais regardez surtout ce qui se passe dans la fenêtre sur laquelle vous avez lancé le serveur web intégré de PHP :



```
D:\Tools\xampp\htdocs>php -S localhost:8000
PHP 7.1.6 Development Server started at Sun Jul 23 14:46:39 2017
Listening on http://localhost:8000
Document root is D:\Tools\xampp\htdocs
Press Ctrl-C to quit.
[Sun Jul 23 14:50:36 2017] ::1:61686 [200]: /dashboard/
[Sun Jul 23 14:50:36 2017] ::1:61687 [200]: /dashboard/stylesheets/normalize.css
[Sun Jul 23 14:50:36 2017] ::1:61692 [200]: /dashboard/stylesheets/all.css
[Sun Jul 23 14:50:36 2017] ::1:61691 [200]: /dashboard/javascripts/modernizr.js
[Sun Jul 23 14:50:36 2017] ::1:61690 [200]: /dashboard/javascripts/all.js
[Sun Jul 23 14:50:36 2017] ::1:61689 [200]: /dashboard/images/xampp-logo.svg
[Sun Jul 23 14:50:36 2017] ::1:61695 [200]: /dashboard/images/fastly-logo.png
[Sun Jul 23 14:50:36 2017] ::1:61694 [200]: /dashboard/images/bitnami-xampp.png
[Sun Jul 23 14:50:36 2017] ::1:61696 [200]: /dashboard/images/social-icons.png
[Sun Jul 23 14:50:36 2017] ::1:61697 [200]: /dashboard/images/favicon.png
[Sun Jul 23 14:50:36 2017] ::1:61698 [200]: /dashboard/images/fastly-logo@2x.png
[Sun Jul 23 14:50:43 2017] ::1:61710 Invalid request (Unexpected EOF)
[Sun Jul 23 15:53:34 2017] ::1:61851 Invalid request (Unexpected EOF)
```

On voit que tous les fichiers CSS, Javascript et PNG qui composent la page du dashboard de Xampp, sont chargés via le serveur web intégré de PHP. Ce serveur web intégré constitue une alternative à Apache. Il peut être lancé à partir de n'importe quel répertoire dans lequel se trouvent des scripts PHP que vous souhaitez exécuter.

Le serveur web intégré offre plusieurs options intéressantes, voici quelques liens pour approfondir le sujet :

<http://php.net/manual/fr/features.commandline.webserver.php>

<https://www.grafikart.fr/tutoriels/php/serveur-web-interne-778>

2.3.3 Environnements virtualisés

Une alternative au fait d'installer un stack PHP sur son PC consiste à monter un stack PHP dans une VM (Virtual Machine) sous Linux. Pour créer une VM, le logiciel Virtual Box d'Oracle est un incontournable (et en plus il est gratuit) :

<https://www.virtualbox.org/>

Le fait de travailler avec une VM offre plusieurs avantages :

- On évite de « polluer » son système d'exploitation avec un environnement PHP, ce dernier étant circonscrit dans la VM, il est invisible pour la machine hôte
- La VM étant sous Linux, elle est généralement plus proche d'un environnement de production (même des différences de configuration peuvent subsister). Les tests effectués dans la VM sont donc plus proches du contexte réel d'utilisation de l'application
- Cela permet aussi de se familiariser avec l'environnement Linux, même si sa machine tourne sous Windows

La création d'une VM n'est pas une opération très compliquée, mais cela prend du temps.

On trouve de nombreux tutos sur le web, ainsi que des vidéos sur Youtube, expliquant comment configurer une VM et y installer un stack PHP.

En matière de VM, une petite révolution s'est produite en 2008, avec l'apparition du projet Vagrant, développé par Mitchell Hashimoto, fondateur de la société HashiCorp. Vagrant est un logiciel libre et open-source pour la création et la configuration d'environnements de développement virtuel. Il peut être considéré comme un wrapper autour de logiciels de virtualisation comme VirtualBox ou VMWare. Il permet surtout d'automatiser la production de VM, au travers de fichiers de configuration que les développeurs ou devops peuvent se passer entre eux, pour produire des VM facilement reproductibles et homogènes.

Pour approfondir le sujet :

<https://www.grafikart.fr/tutoriels/hebergement/vm-vagrant-chef-solo-482>

<https://www.synbioz.com/blog/vagrant et la virtualisation pour faciliter le developpement>

<https://leanpub.com/vagrantcookbook>

<https://leanpub.com/hello-dev-environments>

A partir de 2013, un nouvel outil est venu compléter l'arsenal des développeurs et des administrateur système, j'ai nommé Docker. Ce projet a été développé par Solomon Hykes et son équipe, pour les besoins de leur société dotCloud, société spécialisée dans la mise à disposition de plateforme de type PAAS (platform as a service). A partir du moment où Docker a

été mis à disposition en open source, le projet a rencontré un succès considérable, et de nombreux acteurs du « cloud » se sont mis à proposer des containers s'appuyant sur Docker.

Plus concrètement, Docker permet la mise en œuvre de conteneurs (en anglais « containers ») isolés. Pour comprendre l'intérêt, il faut rapprocher cela du principe des VM.

Le gros défaut des VM traditionnelles (de type VirtualBox), c'est le fait qu'elles sont complètement étanches et ne partagent aucune ressource matérielle (mémoire, disque dur, etc...). Il est dès lors difficile de faire « tourner » plusieurs VM en même temps sur une machine sans mettre cette dernière « à genoux ».

Les containers Docker, au contraire, sont en mesure de partager les ressources d'une même machine, ou d'une même VM (comme la mémoire, le disque dur, etc...), les capacités des serveurs sont ainsi démultipliées. On peut utiliser Docker sur son propre ordinateur pour créer un container dédié au développement PHP.

Quelques articles et livres pour approfondir le sujet :

<https://semaphoreci.com/community/tutorials/dockerizing-a-php-application>

<https://www.newmediacampaigns.com/blog/docker-for-php-developers>

<https://leanpub.com/dockerfordevs>

<http://www.aioptify.com/top-docker-books.php>

2.4 Choisir un outil de test

Nous n'allons pas tarder à programmer, et nous allons faire beaucoup de tests manuels.

Nous verrons que certains tests peuvent être laborieux à réaliser, particulièrement si l'on est amené à les réaliser souvent.

Et il y a un test que nous allons effectuer souvent, c'est celui consistant à lancer des requêtes HTTP (et à vérifier leur résultat), et c'est quelque chose que le logiciel Postman fait très bien. Nous en reparlerons en temps et en heure, mais vous pouvez d'ores et déjà le télécharger et l'installer (ou vous pouvez le faire plus tard, quand nous aborderons le sujet) :

<https://www.getpostman.com/>



J'ai pour objectif de montrer comment utiliser Postman, dans le cadre du développement PHP, mais cet objectif n'est pas atteint dans la version 1.2 de ce support, faute de temps. J'espère trouver le temps de traiter ce sujet dans la version 1.3, à suivre donc...

TODO : à compléter ultérieurement

3 Découverte du langage

AVERTISSEMENT : dans les nombreux exemples qui vont suivre, j'utilise le langage PHP pour générer du code HTML (et aussi dans quelques cas, du CSS et du Javascript). Si vous ne vous sentez pas à l'aise avec le HTML, ou si vous ne savez tout simplement pas de quoi il s'agit, il me semble judicieux que vous vous penchiez sur le support de cours « HTML5 – Premiers pas » qui se trouve dans le dépôt Github suivant :

<https://github.com/gregja/JSCorner>

3.1 Test du stack PHP

Je pars du principe que vous avez installé un stack PHP sur votre ordinateur. Pour ma part, j'utilise Xampp. La plupart des informations qui vont suivre sont valables avec les autres stacks PHP. Si certaines informations sont très spécifiques à Xampp, je le préciserai au fil de l'eau.

Pour vérifier si mon stack PHP fonctionne, c'est facile, je saisis cette URL dans mon navigateur :
<http://localhost/>

On voit que l'URL est aussitôt modifiée pour devenir :

<http://localhost/dashboard/>

Et dans la fenêtre du navigateur, je vois apparaître la page ci-dessous, dont je ne vous ai mis qu'un extrait :



Welcome to XAMPP for Windows 7.1.6

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

La plupart des stacks PHP fournissent un tableau de bord (en anglais « dashboard ») qui ressemble de près ou de loin à celui de Xampp.

On notera que, si je saisis cette autre URL, cela fonctionne aussi :

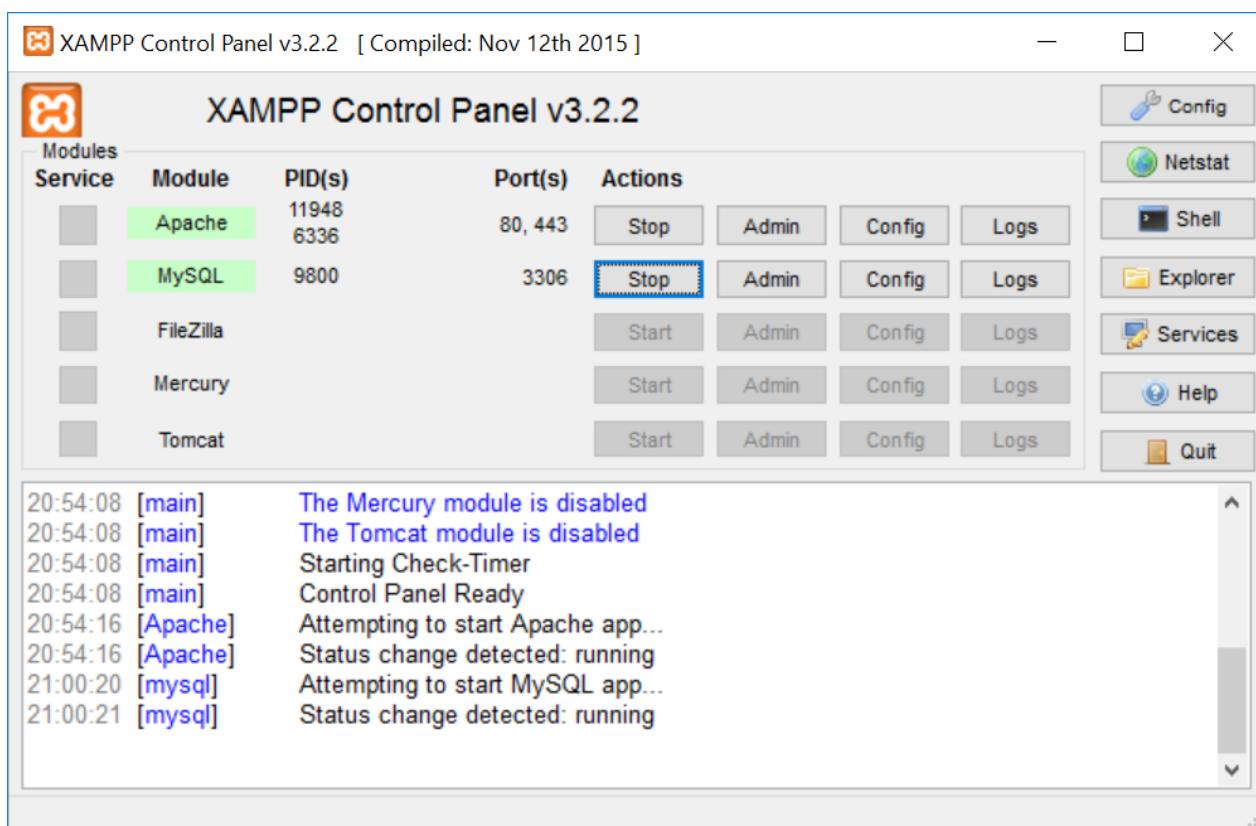
<http://127.0.0.1/>

En effet, dans la configuration du système, l'adresse « localhost » et l'adresse IP « 127.0.0.1 » fonctionnent comme des synonymes. La plupart des systèmes fonctionnent sur ce principe, le vôtre ne devrait pas faire exception.

Tiens, ça ne marche pas sur votre machine ? Que se passe-t-il donc ?

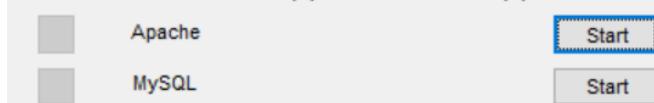
Tout d'abord, est-ce que Xampp est bien démarré ? Vous repérez facilement l'application au moyen de son logo : 

Si l'application Xampp est active, regardez ce qui apparaît dans son « control panel ».



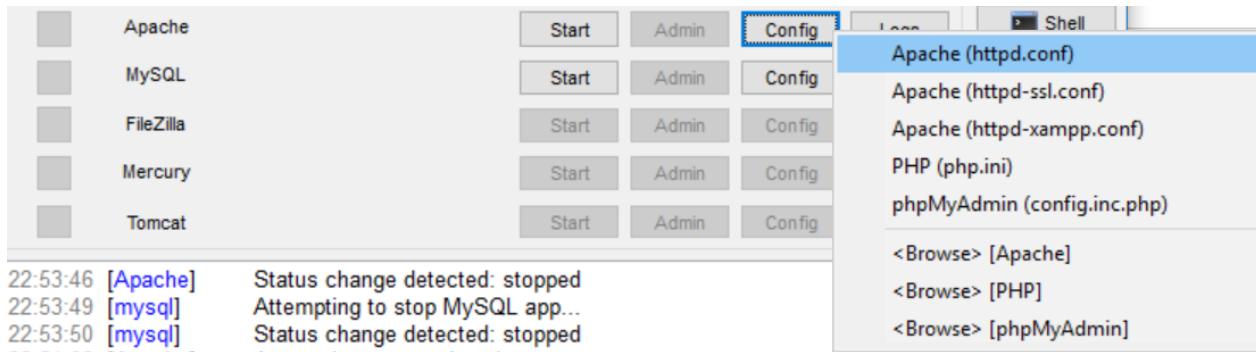
Est-ce que Apache apparaît en vert ? Si oui, tout va bien.

Si Apache apparaît en gris, alors il convient de le démarrer en cliquant sur « Start ». Vous pouvez en profiter pour démarrer aussi MySQL, mais nous n'en aurons pas besoin tout de suite.



Si Apache apparaît en rouge, cela signifie qu'il rencontre un souci au démarrage, il se peut que ce soit dû à un conflit avec un autre logiciel. Une cause possible de conflit, c'est le fait que Apache utilise par défaut le port 80, sur l'adresse IP 127.0.0.1. Si vous avez un autre logiciel qui utilise ce même port sur cette même adresse IP, alors vous avez un conflit.

Si cela arrive, vous pouvez modifier le port en passant par les options de configuration du « control panel » :



Sélectionnez l'option « Apache (httpd.conf) », puis faites défiler l'éditeur jusqu'à la ligne commençant par « Listen 80 » :

The screenshot shows a Windows Notepad window titled "httpd.conf - Bloc-notes". The file contains the Apache configuration file content. The "Listen 80" directive is highlighted in blue. The file also includes several comments (#) explaining the Listen directive.

```

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 80

#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you

```

Modifier la valeur du port, par exemple, vous pouvez remplacer « 80 » par « 81 ». Sauvegardez, fermez l'éditeur, puis essayez de relancer Apache.

Si cela fonctionne, Apache va apparaître en vert.

Dans ce cas, vous pouvez relancer le tableau de bord de Xampp, mais en précisant le port 81 dans l'URL saisie dans le navigateur :

<http://localhost:81/>

Si cela ne fonctionne pas, alors votre problème se situe à un autre niveau, et je vous recommande dans ce cas de lire attentivement la log des erreurs d'Apache (cf. option « Apache error.log »).



Si vous parvenez à identifier une erreur ou un message suspect, parcourez la FAQ (Foire aux Questions) du projet Xampp :

https://www.apachefriends.org/faq_windows.html

... ainsi que le forum sur lequel vous pouvez poser des questions :

<https://community.apachefriends.org/f/>

Attention : avant de poser une question sur un forum, quel qu'il soit, prenez soin de vérifier si la question que vous souhaitez poser n'a pas déjà été posée. Car si c'est le cas, vous risquez de vous faire « incendier » 😞.

Au risque de me répéter, tout ce que je viens d'indiquer sur Xampp est valable avec les autres stacks PHP. Si les « control panels » des autres stacks se présentent différemment, on y retrouve à peu près les mêmes options :

- Pour l'arrêt et le redémarrage d'Apache et de MySQL
- Pour l'accès aux options de configuration (notamment au fichier httpd.conf)
- Pour l'accès aux logs (notamment celles répertoriant les erreurs)

A propos de log, vous avez sans doute remarqué que nous avions deux logs pour Apache (access.log et error.log) et une pour PHP (php_error.log) :

Apache (access.log)

Apache (error.log)

PHP (php_error_log)

Au fait, d'où vient ce terme de « log ». D'après ce que j'ai pu voir sur Wikipédia, un log « serait un terme de marine désignant le livre de bord d'un bâtiment ». Je ne sais pas si le terme utilisé en informatique vient de là, mais un fichier de log, pour un développeur, c'est un fichier d'historique (historique de connexion, comme pour le fichier « access.log », historique des erreurs, comme pour le fichier « error.log »).

Nous utiliserons ces fichiers de log de temps en temps, ils contiennent beaucoup d'informations intéressantes.

3.2 Premier « hello world »

Je pars du principe que votre stack PHP est opérationnel.

Mais au fait, où place-t-on un fichier PHP, au sein du stack PHP, pour pouvoir l'exécuter ?
En fait, l'emplacement diffère légèrement d'un stack à l'autre.

Sur la plupart des stacks PHP, l'emplacement dans lequel on doit déposer les sources PHP est un sous-répertoire du stack lui-même, qui s'appelle « www » ou « htdocs ».

Dans le cas de Xampp, l'emplacement où déposer vos sources PHP est un sous-répertoire « htdocs » se trouvant à l'intérieur du répertoire « Xampp ».

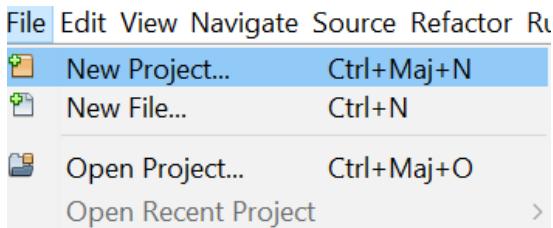
Je vous propose de créer un premier script PHP, tout petit, juste histoire de dire « Bonjour le monde ». Presque tous les développeurs ont démarré l'apprentissage de la programmation par un « hello world », ce serait dommage que vous ratiez ça 😊.

Je vous disais que le répertoire où placer notre code était de type « www » ou « htdocs », selon le stack PHP utilisé. Mais nous n'allons pas placer nos scripts PHP à la racine de ce répertoire, sinon cela va devenir très vite un énorme bazar. Je vous propose donc de créer un sous-répertoire « essaiphp » à l'intérieur de votre répertoire « www » ou « htdocs ». C'est dans ce sous-répertoire « essaiphp » que nous placerons nos premiers scripts PHP. Nous allons considérer que ce sous-répertoire « essaiphp » est notre premier projet PHP. Par la suite, vous pourrez créer d'autres sous-répertoires, sur le même principe.

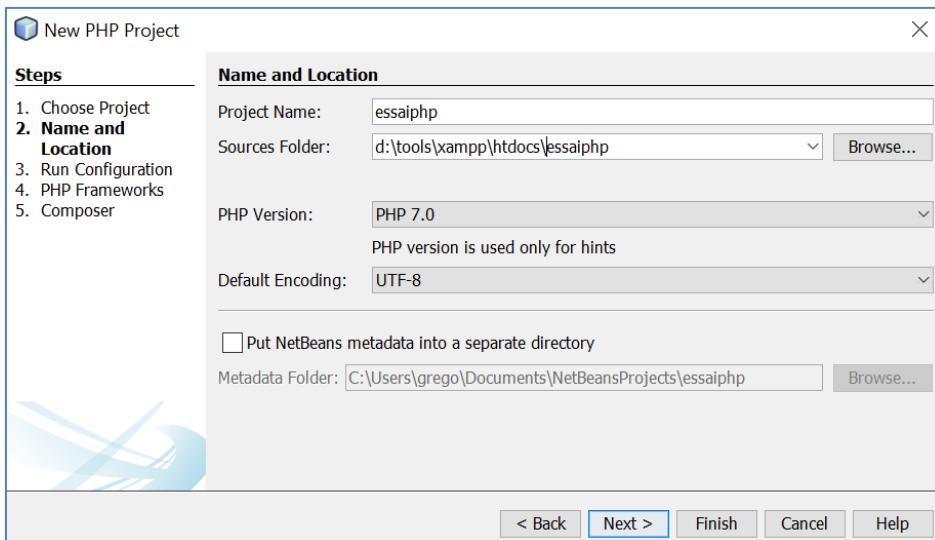
On peut bien évidemment créer le sous-répertoire « essaiphp » à la main, mais nous pouvons aussi le faire au travers d'un éditeur (comme Notepad++ ou SublimeText) ou d'un IDE (comme Netbeans ou Eclipse). Je vous propose de voir comment nous pouvons créer ce répertoire avec Netbeans, vous pourrez facilement transposer la méthode sur d'autres IDE.

Netbeans fonctionne selon une logique de projet. C'est très pratique pour compartimenter les choses, et éviter de mélanger les choux avec les carottes. A chaque projet correspond un répertoire, qui servira de conteneur pour tous les fichiers (scripts PHP, fichiers HTML, CSS, Javascript, etc..) et sous-répertoires du projet.

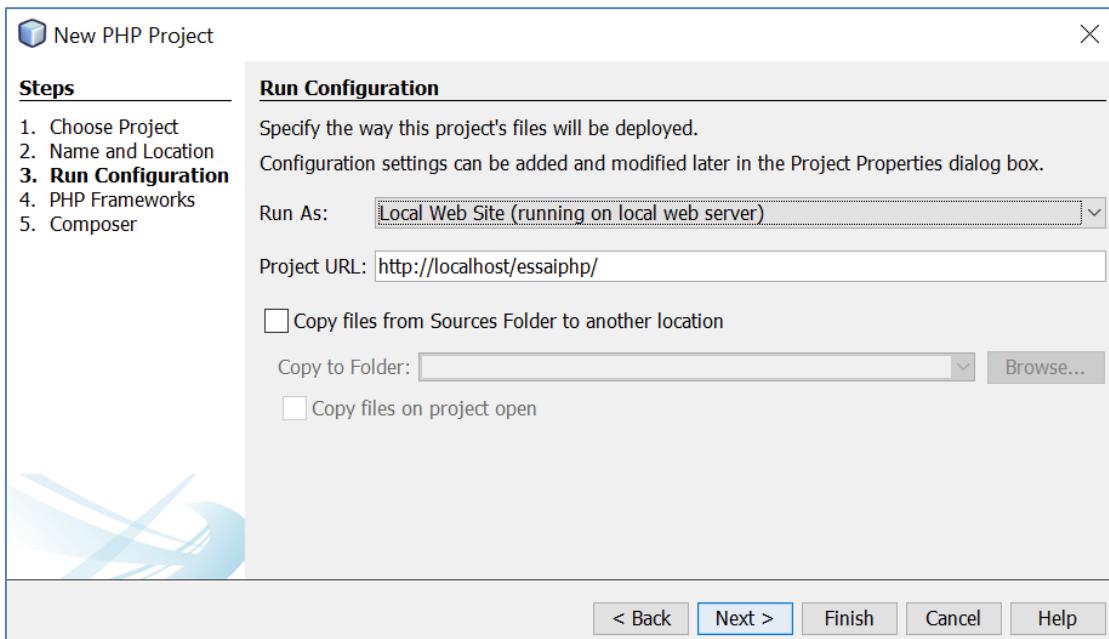
Dans Netbeans, je sélectionne donc l'option « new project » qui se trouve dans le menu « File ».



Puis j'indique le nom du projet, et je modifie le paramètre « sources folder », de manière à pointer sur le sous-répertoire « xampp/htdocs » de mon stack PHP :



Dans la fenêtre qui suit, Netbeans nous indique les paramètres qu'il retient pour la configuration du projet, par rapport au stack PHP en place. Ces paramètres nous permettront de lancer le projet directement à partir de Netbeans :

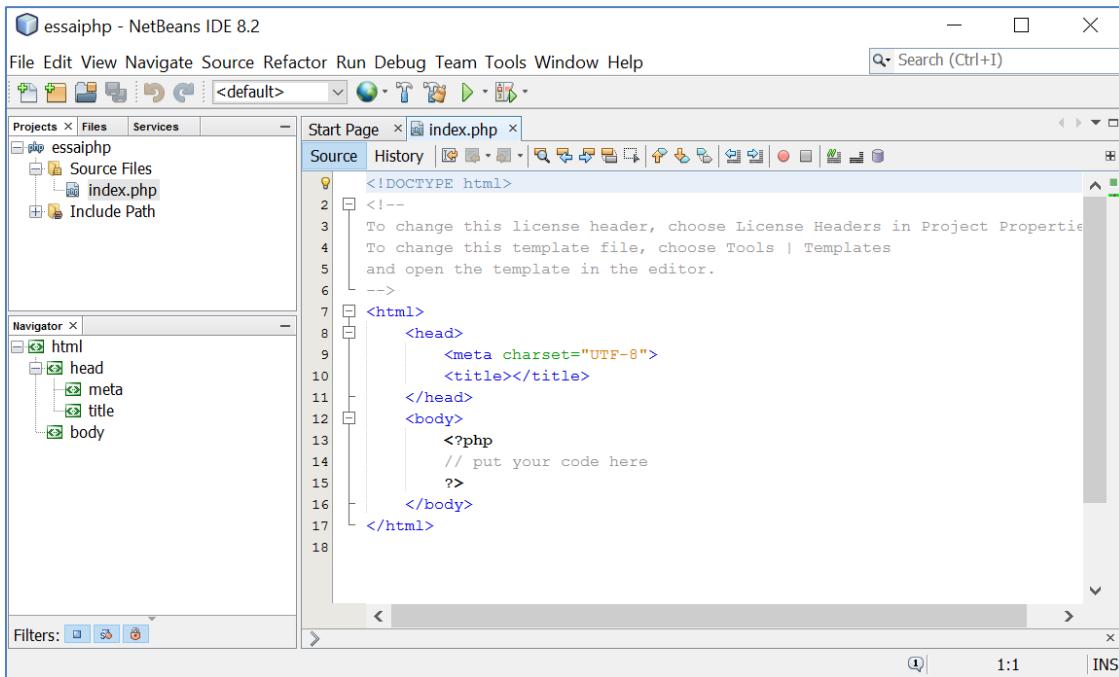


Si vous avez rencontré des conflits au niveau du port, lors de la configuration de votre stack PHP, et que vous avez été obligé de modifier ce port (par exemple en sélectionnant le port 81), alors vous devrez le préciser dans le paramètre Project URL, comme ceci :

<http://localhost:81/essaiphp>

Cliquez sur « Finish », et patientez...

Si tout s'est bien passé, vous devriez obtenir à peu près ceci :



L'écran est découpé en 3 vues. La vue en haut à gauche présente les projets, la vue de droite présente un script PHP qui a été généré automatiquement par Netbeans (il est trop cool le Netbeans), et en bas à gauche, on a une vue « navigator » qui se rapproche beaucoup de la représentation du DOM que nous propose un navigateur web quand on utilise l'inspecteur d'élément.

Le fichier généré automatiquement par Netbeans s'appelle « index.php ». Ce n'est pas innocent, mais je préfère ne pas en parler maintenant. On sait que le fichier généré est un script PHP, du fait de la présence du suffixe « .php » dans le nom du fichier. On reviendra sur ce sujet plus tard également.

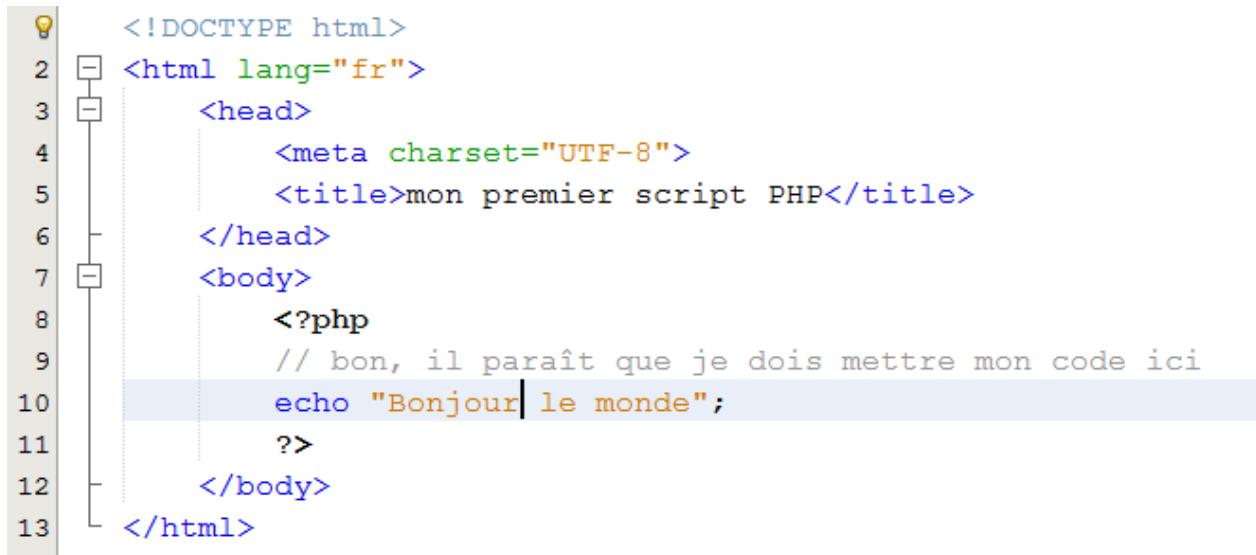
Pour l'instant, je vous demande juste de retenir que le fait que ce fichier soit un script PHP nous autorise à placer du code PHP à l'intérieur.

Pour insérer du code PHP dans un script, on utilise les balises suivantes :

```
<?php
// put your code here
?>
```

Ce sont d'ailleurs ces balises que Netbeans a inséré dans le squelette de script qu'il a généré pour nous. Il les a placées à l'intérieur de la balise « body » du code HTML.

Comme ce script contient des commentaires HTML qui ne servent pas à grand-chose, je vous propose d'élaguer un peu le code pour aboutir au résultat suivant :



```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="UTF-8">
        <title>mon premier script PHP</title>
    </head>
    <body>
        <?php
            // bon, il paraît que je dois mettre mon code ici
            echo "Bonjour le monde";
        ?>
    </body>
</html>
```

Pour vous aider à y voir clair, j'ai effectué les modifications suivantes :

- J'ai ajouté un attribut « lang » sur la balise « html »
- J'ai ajouté un titre dans la balise HTML « title »
- J'ai ajouté l'instruction PHP « echo » à l'intérieur des balises PHP

On peut dès maintenant tester ce script. Pour ce faire, ouvrez votre navigateur préféré, et saisissez l'URL suivante :

<http://localhost/essaiphp>

... ou celle-ci si vous avez changé votre configuration d'Apache pour le port 81 par exemple :

<http://localhost:81/essaiphp>

Si tout s'est bien passé, vous devriez voir apparaître le message suivant dans la fenêtre de votre navigateur :

Bonjour le monde

Si cela ne fonctionne pas, je vous invite à relire le chapitre 3.1 qui je l'espère, vous aidera à identifier l'origine de la panne.

Si vous avez bien obtenu votre message « Bonjour le monde », alors je vous invite à regarder à quoi ressemble le code HTML que le navigateur a reçu. C'est facile, nous l'avons fait souvent dans le cours « HTML5 – Premiers pas », mais je rappelle quand même la méthode : faites un clic-droit sur le fond de la page web, et sélectionnez l'option « affichage du code source ». Vous devriez obtenir ceci :

```

1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8">
5     <title>mon premier script PHP</title>
6   </head>
7   <body>
8     Bonjour le monde      </body>
9 </html>
10

```

Bon, j'espère que ça vous rappelle des souvenirs, parce que dans le cas contraire, il faut vraiment que vous fassiez un break et que vous lisiez le cours « HTML5 – Premier pas ».

Dans le code ci-dessus, vous voyez que les balises PHP ont disparu. Pour bien comprendre ce qui s'est passé, je vous propose de mettre côté à côté le code PHP (à gauche) et le code HTML produit par ce même code PHP (à droite) :

<pre> 1 <!DOCTYPE html> 2 <html lang="fr"> 3 <head> 4 <meta charset="UTF-8"> 5 <title>mon premier script PHP</title> 6 </head> 7 <body> 8 <?php 9 // bon, il paraît que je dois mettre mon code ici 10 echo "Bonjour le monde"; 11 ?> 12 </body> 13 </html> </pre>	<pre> 1 <!DOCTYPE html> 2 <html lang="fr"> 3 <head> 4 <meta charset="UTF-8"> 5 <title>mon premier script PHP</title> 6 </head> 7 <body> 8 Bonjour le monde </body> 9 </html> 10 </pre>
--	---

Quand l'interpréteur PHP démarre l'exécution du script « index.php », il commence par ouvrir un « buffer ». Ce terme barbare – le « buffer » - désigne une mémoire tampon, dans laquelle l'interpréteur va envoyer tout le texte qu'on lui demande de produire en sortie.

Parmi les éléments que l'interpréteur envoie dans ce « buffer », on trouve :

- le code HTML « brut », tel qu'il est contenu dans le script PHP, du moment qu'il est placé en dehors des balises PHP : c'est le cas dans notre exemple de la majeure partie de l'entête de la page HTML, et des balises de fermeture « /body » et « /html ».
- Le code se trouvant entre les balises PHP va être analysé et exécuté, et si des instructions telles que « echo » ou « print » s'y trouvent, elles vont envoyer, elles aussi, du texte dans le buffer HTML.

Fort de toutes ces explications, je vous invite à comparer attentivement les 2 codes (PHP et HTML) dans la page qui précède.

Tiens, vous avez sans doute remarqué, que dans le code source HTML généré, la balise « /body » se trouve placée bizarrement. Cela n'est pas très grave, car le code HTML est correct, mais c'est vrai que ça fait bizarre. On peut corriger cela facilement en modifiant légèrement l'instruction « echo ». Essayez ceci :

```
<?php  
// bon, il paraît que je dois mettre mon code ici  
echo "Bonjour le monde" . PHP_EOL ;  
?>
```

Explication : le point placé entre « Bonjour le monde » et PHP_EOL, c'est le symbole de concaténation en PHP. Je rappelle qu'en Javascript on utilise un « + », mais en PHP c'est un « . ». C'est comme ça, ne me demandez pas pourquoi, je n'y suis pour rien ☺. Pour les grands débutants, je rappelle que le principe de la concaténation, c'est de coller bout à bout des informations pour produire une chaîne de caractères.

Le mot clé PHP_EOL, c'est une constante PHP. Le « EOL » correspond à « End Of Line », soit en français « fin de ligne ». Cette constante a donc pour effet de générer un caractère de saut de ligne. Vous ne le savez probablement pas, mais cette constante PHP_EOL nous simplifie la vie, car le caractères de saut de ligne n'est pas le même selon les systèmes d'exploitation :

\r = CR (Carriage Return) // saut de ligne sur Mac OS, avant la génération OS X
\n = LF (Line Feed) // saut de ligne sur Linux/Unix/Mac OS X
\r\n = CR + LF // saut de ligne sur Windows

Ce n'est pas stratégique, mais c'est intéressant de connaître la signification de ces caractères :

- « line feed » signifie en anglais « saut de ligne »
- « carriage return » signifie « retour chariot » : le retour chariot, c'est une opération manuelle que l'on devait effectuer sur les vieilles machines à écrire mécanique, chaque fois que l'on souhaitait changer de ligne. Je dois dire que c'est vraiment amusant de taper du texte sur une machine à écrire mécanique. Si vous n'avez jamais eu l'occasion d'en voir fonctionner, je vous recommande de visionner le film "Populaire" de Régis Roinsard, avec Romain Duris en tête d'affiche (le film est sorti en 2012).

Pourquoi sous Windows faut-il utiliser le double jeu de caractères \r\n ? A priori, il s'agissait pour Windows de maintenir une compatibilité avec les vieux systèmes d'exploitation MS-DOS et CP/M. Si le sujet vous intéresse, vous pouvez trouver pas mal d'infos sur cette page :

<https://softwareengineering.stackexchange.com/questions/29075/difference-between-n-and-r-n>

Donc, en résumé, la constante PHP_EOL nous simplifie la vie car on n'a pas besoin de se préoccuper de savoir quel jeu de caractère utiliser pour générer un saut de ligne sur notre environnement PHP (PHP le gère pour nous au travers de cette constante).

Sauvegardez votre modification et rafraîchissez la page dans le navigateur. En façade, ça n'a rien changé, mais dans les coulisses, le code HTML est mieux formaté :

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8">
5     <title>mon premier script PHP</title>
6   </head>
7   <body>
8     Bonjour le monde
9   </body>
10 </html>
```

Voilà, vous venez de découvrir la concaténation en PHP, plus une première constante PHP (PHP_EOL). Nous en découvrirons d'autres au fil de l'eau.

Dans les chapitres qui suivent, nous allons continuer avec notre « hello world », mais avant ça, je vais vous montrer quelques astuces sympas dans les 2 prochains chapitres.

3.3 Quelques astuces à connaître

3.3.1 Astuces d'un haricot du net

Nous allons parler ici de quelques astuces propres à Netbeans.

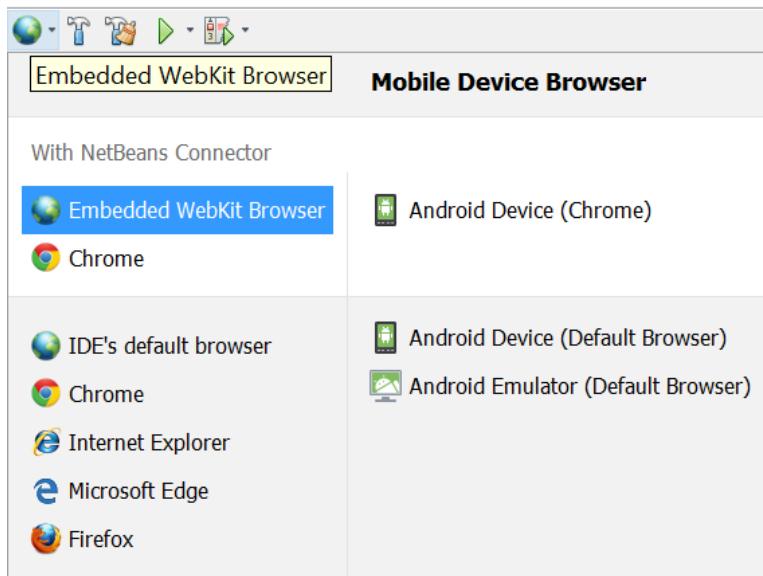
Il y a quelques instants, je vous invitais à lancer votre navigateur et à taper l'URL de votre projet « essaiphp ».

En fait, vous disposez d'un raccourci dans Netbeans pour lancer votre application.

Pour cela, il vous suffit de cliquer sur l'icône en forme de mappemonde, qui correspond à un moteur de rendu HTML, qui s'appelle WebKit, et qui est intégré à Netbeans. WebKit est un moteur très utilisé, et vous pouvez trouver des informations à son sujet sur Wikipédia :

<https://fr.wikipedia.org/wiki/WebKit>

Vous n'êtes pas limité à WebKit, vous pouvez aussi sélectionner Chrome, Firefox, IE, Edge... donc à peu près tous les navigateurs qui sont embarqués sur votre PC.



Sélectionner un navigateur ne suffit pas, il faut ensuite cliquer sur l'icône ➡ pour déclencher l'exécution du script sélectionné. Si vous avez demandé le « Embedded WebKit Browser », alors la page s'affiche dans une vue de Netbeans, dans le cas contraire le navigateur que vous avez sélectionné est « appelé » automatiquement.

3.3.2 Index ou pas index ?

Peut être vous êtes-vous demandé pourquoi en saisissant l'URL suivante :

<http://localhost/essaiphp>

... on lance le script index.php.

En fait, si on avait utilisé l'URL suivante, on aurait abouti au même résultat :

<http://localhost/essaiphp/index.php>

Mais par quelle magie ? En fait, Apache est configuré d'une manière particulière :

- Si on lui demande d'exécuter un script particulier à l'intérieur du répertoire « essaiphp », Apache fait le job
- Si on n'indique pas à Apache quel script on souhaite exécuter à l'intérieur de « essaiphp », alors il recherche si un script « index.php » existe dans ce même répertoire. Si c'est le cas, il demande à l'interpréteur PHP de l'exécuter, et il envoie au navigateur la réponse produite par le PHP. Si ce n'est pas le cas, il recherche s'il existe un fichier « index.html ». Si c'est le cas, il l'envoie au navigateur.

Et s'il n'existe pas non plus de fichier « index.html » ? Que se passe-t-il ?

... je vous propose de faire l'expérience. Dans votre IDE, faites un clic-droit sur le fichier « index.php » et sélectionnez l'option « rename ». Changez le nom du fichier en « helloworld.php », puis validez.

Rafraîchissez la page de votre navigateur qui pointe sur l'URL suivante :

<http://localhost/essaiphp>

Que remarquez-vous ?

Index of /essaiphp			
<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 helloworld.php	2017-07-07 14:07	292	
 nbproject/	2017-07-07 00:09	-	

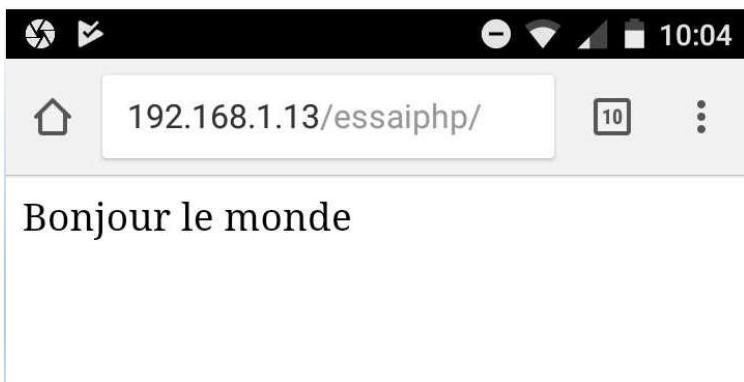
Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.1.6 Server at localhost Port 80

La configuration d'Apache sur votre environnement de développement est ainsi faite que

Apache vous propose une vue sur le contenu du répertoire. Ce comportement très pratique sur un environnement de développement n'est pas souhaitable sur un environnement de production, pour des raisons de sécurité (on n'a pas envie qu'une application donne aussi facilement aux pirates des informations sur le contenu de ses répertoires). Du coup, sur une application de production, Apache sera généralement configuré pour renvoyer, soit une page blanche, soit un message d'erreur indiquant que le répertoire n'est pas accessible. Bref, l'Apache de votre stack PHP de développement est bavard comme un vieil indien qui n'aurait pas vu passer de caravane depuis longtemps, et qui taperait la « discute » avec le premier venu. Pas grave, puisqu'en environnement de développement, ce comportement « bavard » nous arrange.

3.4 Mon « hello world » dans mon smartphone

Vous avez vu votre page « hello world » fonctionner avec le navigateur de votre PC... Cela vous dirait de voir cette même page s'afficher dans le navigateur de votre smartphone ? Vous ne pensiez pas que c'était possible ? Pourtant je vous garantis que vous pouvez le faire :

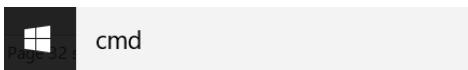


Bon, pour y arriver, il y a un peu de configuration à faire.

En premier lieu, il faut que votre PC (sur lequel « tourne » le stack PHP) et votre smartphone soient connectés à la même borne Wi-fi.

Ensuite vous avez besoin de connaître l'adresse IP de votre PC, car c'est elle que nous allons indiquer dans l'URL du navigateur du smartphone. Par exemple, mon adresse IP ici est 192.168.1.13. Si on mettait « localhost » dans le navigateur du smartphone, on tournerait en rond à l'intérieur du smartphone..., donc il nous faut l'adresse de la machine distante (la machine « remote », comme disent nos amis anglophones).

Pour connaître l'adresse IP de notre PC, nous devons utiliser la ligne de commande du PC. Pour lancer la ligne de commande, nous allons utiliser la commande « cmd » :

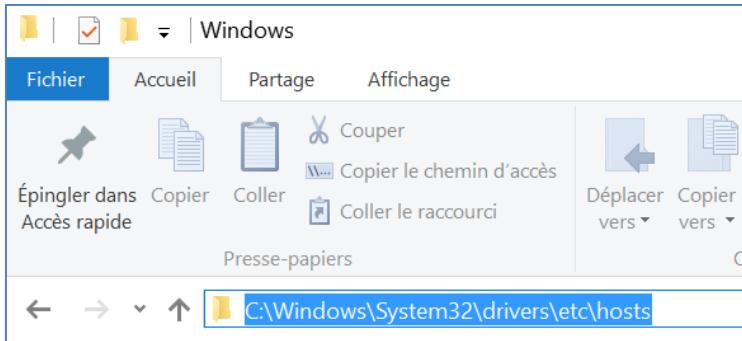


Dès que la fenêtre « invite de commandes » apparaît, saisissez la commande « ipconfig » (puis pressez la touche « Entrée »).

Comme notre connexion se fait via le Wi-fi, c'est l'adresse IP qui se trouve dans la partie « carte réseau sans fil Wi-Fi » qui nous intéresse :

Notez précieusement cette adresse IP, car on va en avoir besoin en fin de parcours.

Vous pouvez d'ores et déjà faire un essai de connexion sur le navigateur de votre smartphone, car selon la configuration de votre PC, il est possible que cela fonctionne tout de suite. Mais si ça ne fonctionne, il faut jeter un coup d'œil à votre fichiers « hosts ». Ce fichier est assez facile à trouver :



C:\Windows\System32\drivers\etc\hosts

```

1 # Copyright (c) 1993-2009 Microsoft Corp.
2 #
3 # This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
4 #
5 # This file contains the mappings of IP addresses to host names. Each
6 # entry should be kept on an individual line. The IP address should
7 # be placed in the first column followed by the corresponding host name.
8 # The IP address and the host name should be separated by at least one
9 # space.
10 #
11 # Additionally, comments (such as these) may be inserted on individual
12 # lines or following the machine name denoted by a '#' symbol.
13 #
14 # For example:
15 #
16 #      102.54.94.97      rhino.acme.com          # source server
17 #              38.25.63.10      x.acme.com            # x client host
18 #
19 # localhost name resolution is handled within DNS itself.
20 #      127.0.0.1      localhost
21 #              ::1      localhost
22

```

A la fin du fichier « hosts », ajoutez les 2 lignes en gris ci-contre.

Au moment de sauvegarder vos modifications, il y a de grandes chances pour que votre système vous dise que votre éditeur n'a pas le niveau d'autorisation nécessaire pour modifier le fichier « hosts ». Si c'est le cas, il devrait vous proposer de rouvrir l'éditeur en mode « administrateur »... faites-lui plaisir, répondez « oui ». Cela réglera votre problème.

Après toute modification du fichier « hosts », il est recommandé de rebooter, pour que la modification soit prise en compte par le système.

A partir de là, il n'y a plus qu'à lancer le navigateur du smartphone, et à croiser les doigts :



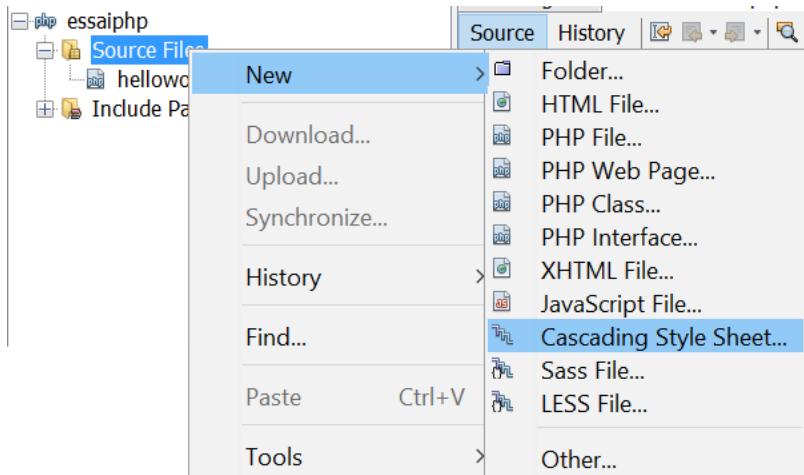
Désormais, vous êtes en mesure de tester tous vos développements sur votre PC, mais aussi sur tout appareil distant (smartphone, tablette, autre PC) qui se trouve connecté à la même borne Wi-Fi. Plutôt cool, non ?

Bon, mais il est toujours aussi moche ce « hello world »... on se le refait ?

3.5 Second « hello world » avec du style

Pour donner du style à notre « hello world », nous devons passer par du CSS.

Je vous propose d'ajouter un fichier CSS à notre projet « essaiphp », que nous appellerons « helloworld.css ». Pour cela, un clic-droit dans Netbeans, et le tour est joué :



Dans le fichier CSS créé, saisissez le code suivant :

```
.annonce {  
    font-size: 5em;  
    text-align: center;  
    color: lightblue;  
    text-shadow: 2px 2px 4px #000000;  
    width: 50%;  
    margin: auto;  
}
```

Modifiez le code du script PHP, ou mieux, copiez-le et créez un nouveau script « helloworld2.php ». Voici son code :

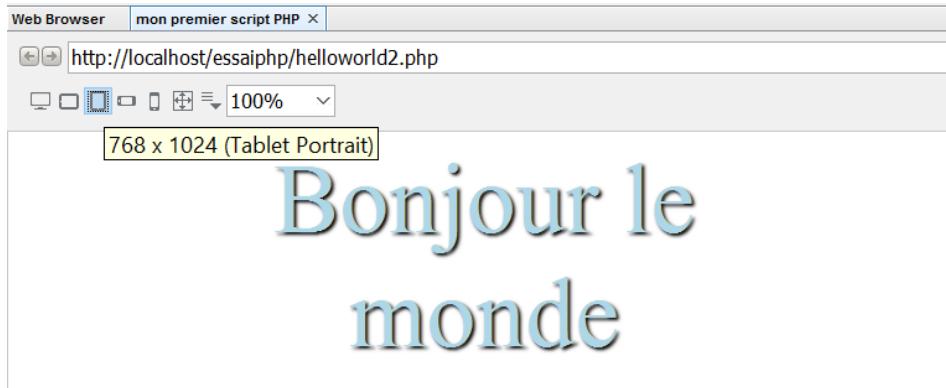
```
<!DOCTYPE html>  
<html lang="fr">  
    <head>  
        <meta charset="UTF-8">  
        <title>mon premier script PHP</title>  
        <link rel="stylesheet" href="helloworld.css" media="screen">  
    </head>  
    <body>  
        <div class="annonce">  
            <?php  
                // bon, il paraît que je dois mettre mon code ici  
                echo "Bonjour le monde" . PHP_EOL ;  
            ?>  
        </div>  
    </body>  
</html>
```

Ah, là c'est vraiment la classe !!! 😊

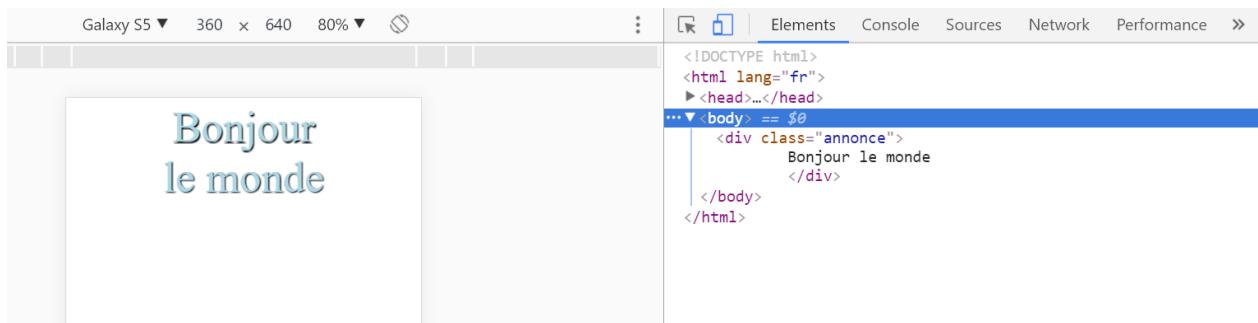


Bonjour le
monde

Une autre petite astuce à noter au sujet de Netbeans : si vous utilisez le WebKit, vous remarquerez qu'il vous offre une série de boutons permettant de changer la résolution de la vue de simulation :



Je rappelle aussi un point évoqué en annexe du cours « CSS3 – Premiers pas » : le navigateur Chrome est doté d'un simulateur de smartphone, que l'on peut activer en passant par les outils de développement :



Mais nous avons vu dans le chapitre précédent, que nous pouvons tester notre application via notre propre smartphone, et ça c'est vraiment cool !!! 😊

3.6 Troisième « hello world » pour découvrir les variables

Notre précédent « hello world » était très basique. Nous avions 2 blocs de code HTML, et un bloc de code PHP glissé entre les deux. En fait, nous avons la possibilité de placer plusieurs blocs de PHP, disséminés dans le code HTML. Nous pouvons par exemple avoir un bloc de code, placé au tout début du script, et qui sert à initialiser des variables. Il pourrait ressembler à ceci :

```
<?php
    $titre = 'Mon 3ème Helloworld';
    $encodage = 'UTF-8';
    $message1 = "Bonjour le monde";
    $message2 = "jouons avec les variables";
    $taille_message2 = 28;
?>
```

Les variables en PHP sont toujours préfixées par un dollar.

Dans l'exemple ci-dessus, nous avons déclaré 4 variables de type chaîne de caractères (en anglais : « string »), et une variable de type nombre entier (en anglais : « integer »).

Vous remarquerez qu'une chaîne de caractères peut être délimitée par des apostrophes ou des guillemets, nous y reviendrons plus tard.

Nous pouvons disséminer les variables un peu partout dans notre code HTML, à condition de toujours les encadrer par les balises PHP. Voici un exemple de ce que cela peut donner :

```
<?php
    $titre = 'Mon 3ème Helloworld';
    $encodage = 'UTF-8';
    $message1 = "Bonjour le monde";
    $message2 = "jouons avec les variables";
    $taille_message2 = 28;
?><!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="<?php echo $encodage; ?>">
        <title><?php echo $titre; ?></title>
        <link rel="stylesheet" href="helloworld.css" media="screen">
    </head>
    <body>
        <div class="annonce">
            <?php
                // bon, il paraît que je dois mettre mon code ici
                echo $message1 . PHP_EOL ;
                echo '<br>' . PHP_EOL ;
            ?>
            <span style="font-size:<?php echo $taille_message2; ?>px">
                <?php
                    echo $message2 . PHP_EOL ;
                ?>
            </span>
        </div>
    </body>
</html>
```

Bon, si vous trouvez que le script PHP de la page précédente est un vrai bazar, je ne peux pas vous contredire. Dans les débuts de PHP, la plupart des scripts PHP ressemblaient à ça. Mais assez rapidement, certaines techniques ont été utilisées pour mieux structurer le code des scripts, et éviter d'aboutir à ce joyeux bazar (et nous allons étudier certaines de ces techniques au fil de l'eau).

Regardons ce que cela donne dans le navigateur :



Nous pouvons demander au navigateur de nous montrer le code source qu'il a reçu :

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8">
5     <title>Mon 3ème HelloWorld</title>
6     <link rel="stylesheet" href="helloworld.css" media="screen">
7   </head>
8   <body>
9     <div class="annonce">
10       Bonjour le monde
11     <br>
12     <span style="font-size:28px">
13       jouons avec les variables
14     </span>
15   </div>
16 </body>
17 </html>
18
```

Vous voyez que toutes les variables PHP ont été remplacées par leurs valeurs respectives, il ne reste dans le code source HTML aucune trace du code PHP d'origine. L'interpréteur PHP a donc bien fait son travail.

3.7 Quatrième « hello world », avec un peu de JS

Je vous propose de copier le script « helloworld3.php » pour créer un « helloworld4.php ». L'objectif ici, c'est de voir comment se passe l'inclusion de code Javascript dans une page HTML générée via un script PHP.

Nous allons créer un fichier Javascript dans notre répertoire/projet « essaiphp ». Je vous propose de l'appeler « helloworld.js » dont voici le code source :

```
var span_a_cacher = document.querySelector('span');
span_a_cacher.addEventListener('click', function(evt) {
    evt.target.style = "display:none";
}, false);
```

C'est vraiment un code hyper basique, nous avons étudié des choses plus sophistiquées que cela dans le cours « Javascript – Premier pas ». En substance, nous cherchons dans le DOM une balise « span ». Nous prenons la première que nous trouvons et nous lui affectons un écouteur d'événement de type « click » qui va avoir pour effet de faire disparaître la balise « span » et son contenu, dès que nous cliquerons dessus.

Dans notre script « helloworld4.php », ajoutons la ligne ci-dessous juste avant la balise « /body » :

```
<script src="helloworld.js"></script>
```

Il ne reste plus qu'à tester :

Bonjour le monde

jouons avec les variables

Cliquez sur « jouons avec les variables ». Si vous n'avez pas commis d'erreur, cette « span » doit disparaître.

Bonjour le monde

3.8 Dans les coulisses d'un « hello world »

Avant de nous attaquer à une étude plus approfondie du PHP, ce que nous ferons dans la suite de ce cours, je vous propose de faire une pause, et de regarder plus précisément ce qui se passe quand vous saisissez une URL dans le navigateur, et que cette URL a pour effet de déclencher l'exécution d'un script PHP. Pour ce faire, nous allons étudier en profondeur ce qui se passe dans les coulisses de notre quatrième « hello world ».

Quand je saisis dans mon navigateur l'URL suivante :

<http://localhost/essaiphp/helloworld4.php>

... le navigateur déclenche une requête HTTP vers le serveur qui a pour nom « localhost ». Dans une configuration « locale » comme la nôtre, ce « localhost » correspond à l'adresse IP « 127.0.0.1 ».

Si l'URL fait référence à un nom autre que localhost, comme par exemple :

<https://www.meetup.com>

... cette adresse « meetup.com » n'étant pas définie comme une adresse locale sur mon PC, une requête HTTP va être lancée sur le réseau internet, pour rechercher le serveur auquel correspond l'URL demandée.

Notre environnement web local est assez proche d'un environnement web véritable, si ce n'est que tout se passe à l'intérieur de ma machine.

Pour espionner le déroulement des opérations côté navigateur, je vous propose d'activer les outils de développement du navigateur (touche F12 sur certains navigateurs), et d'ouvrir l'onglet « réseau » (ou « network » selon les navigateurs).

Si vous êtes sur Firefox, l'onglet « réseau » est matérialisé par cette icône : 

Commençons par demander au navigateur d'afficher une page avec une URL qui n'existe pas sur notre serveur, comme par exemple :

<http://localhost/essaiphp/helloworldx.php>

Voici ce que nous indique Firefox :

The screenshot shows the Firefox developer tools Network tab. A red box highlights the row for the failed request to 'helloworldx.php'. The status column shows '404' and the type column shows 'GET'. The URL is listed as 'http://localhost/essaiphp/helloworldx.php'. The response time is shown as '210 ms'.

Et voici le résultat pour la même URL dans Chrome :

The screenshot shows the Chrome developer tools Network tab. A red box highlights the row for the failed request to 'helloworldx.php'. The status column shows '404' and the type column shows 'doc...'. The URL is listed as 'http://localhost/essaiphp/helloworldx.php'. The response time is shown as '5 ...' ms.

En tapant l'URL précédente, nous avons lancé une requête HTTP qui a été dirigée vers le serveur web local piloté par notre stack PHP. Ce serveur web a renvoyé une réponse, qui apparaît ici en rouge.

Si l'affichage diffère un peu, on retrouve les mêmes informations, dont une doit vous sauter aux yeux : la colonne « statut » ou « état », indique le code 404. Ce code 404 correspond à l'un des codes retour qu'un serveur web peut renvoyer en réponse à une requête HTTP.

Mais c'est quoi HTTP ?

Le sigle HTTP signifie « Hypertext Transfer Protocol ». Il s'agit d'un « protocole de communication », qui s'appuie sur une norme définissant la manière dont les ordinateurs communiquent sur le réseau internet. Vous pouvez trouver une présentation détaillée sur Wikipédia, que je vous invite à lire (quand vous aurez le temps, ce n'est pas urgent) :

https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Pour l'heure, nous nous avons juste besoin de savoir à quoi correspond ce code retour 404.

Il existe plusieurs codes retours, mais les plus importants (à connaître) sont les suivants :

- Code retour 200 : la ressource demandée par la requête est disponible, la requête HTTP a bien abouti et le navigateur a reçu la réponse attendue
- Code retour 404 : la ressource demandée n'est pas disponible (elle a peut être été supprimée, ou peut être n'a-t-elle jamais existé), la requête HTTP n'a pas abouti et le navigateur n'a pas reçu de réponse qu'il soit en mesure de traiter

OK, lançons maintenant la requête correspondant à notre 4^{ème} helloworld :

<http://localhost/essaiphp/helloworld4.php>

Voici ce que ça donne dans Firefox :

The screenshot shows the Firefox Developer Tools Network tab. On the left, the page content displays "Bonjour le monde" and "jouons avec les variables". The Network tab lists three requests:

Etat	Méth...	Fichier	Domaine	So...	Ty...	Tr...	Tai...
200	GET	helloworld4.php	localhost	docu...	html	443 o	443 o → 12 ms
200	GET	helloworld.css	localhost	styl...e...	css	mis en ...	153 o
200	GET	helloworld.js	localhost	script	js	mis en ...	345 o

On the right, the Timeline tab shows the execution of three GET requests:

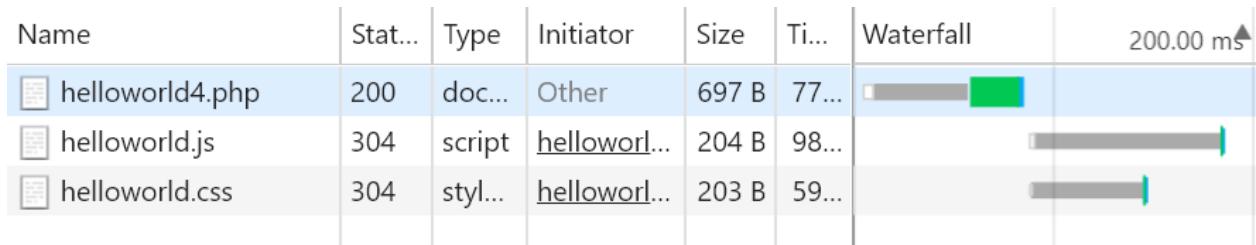
- GET <http://localhost/essaiphp/helloworld4.php> [HTTP/1.1 200 OK 12 ms]
- GET <http://localhost/essaiphp/helloworld.css> [HTTP/1.1 200 OK 0 ms]
- GET <http://localhost/essaiphp/helloworld.js> [HTTP/1.1 200 OK 0 ms]

Et voici la même réponse obtenue dans Chrome :

The screenshot shows the Google Chrome DevTools Network tab. The page content is identical to the Firefox example. The Network tab shows the following requests:

Name	Stat...	Type	Initiator	Size	Ti...	Waterfall	200.00 ms
helloworld4.php	200	doc...	Other	697 B	77...	███████	200.00 ms
helloworld.js	304	script	helloworld...	204 B	98...	███████	200.00 ms
helloworld.css	304	styl...	helloworld...	203 B	59...	███████	200.00 ms

Je vous propose de regarder plus attentivement le petit tableau proposé par Chrome :



Ce tableau est intéressant car il nous montre, dans la partie de droite, une ligne de temps, dans laquelle on voit que le chargement de la page s'est fait en plusieurs temps :

- Temps 1a : le navigateur lance une requête HTTP vers un serveur (dont l'adresse IP n'apparaît pas ici) pour demander à ce qu'on lui envoie la ressource « helloworld4.php ».
- Temps 1b : le navigateur reçoit une réponse avec un code retour « 200 » qui signifie « tout va bien ».
- Temps 1c : le navigateur analyse le contenu de la réponse, il s'agit de code HTML, comme nous le vérifierons dans un instant.
- Temps 2 : Si vous vous souvenez du code produit par notre script « helloworld4.php », vous savez que le code HTML produit par le script PHP fait référence à 2 fichiers externes (« helloworld.js » et « helloworld.css ») qui se trouve dans le même répertoire que le script PHP lui-même. C'est ce que découvre le navigateur en analysant le code HTML renvoyé par la ressource « helloworld4.php ». Du coup le navigateur lance 2 nouvelles requêtes HTTP pour demander au serveur de lui envoyer les ressources « helloworld.js » et « helloworld.css ».

Les étapes que je viens de décrire se sont déroulées en moins d'un 1/5^{ème} de seconde, ce qui explique que vous ayiez l'impression que le chargement de la page est instantané. Et c'est vrai qu'à notre échelle, il est instantané.

Je vous propose de regarder quelques points plus en détail.

Cliquez sur la ligne correspondant à la ressource « helloworld4.php ». Vous voyez apparaître plein d'informations, en fait il s'agit d'une présentation détaillée de la requête HTTP.

Name

- helloworld4.php**
- helloworld.css
- helloworld.js

Headers

Request URL: http://localhost/essaiphp/helloworld4.php

Request Method: GET

Status Code: 200 OK

Remote Address: [::1]:80

Referrer Policy: no-referrer-when-downgrade

Response Headers

Connection: Keep-Alive

Content-Length: 443

Content-Type: text/html; charset=UTF-8

Date: Mon, 10 Jul 2017 09:02:30 GMT

Keep-Alive: timeout=5, max=99

Server: Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.1.6

X-Powered-By: PHP/7.1.6

Request Headers

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8

Accept-Encoding: gzip, deflate, br

3 requests | 696 B transferred ...

Il y a plein d'informations à « gratter » là-dedans, mais hormis quelques points de détail sur lesquels je reviendrai à l'occasion, pour le reste c'est assez barbant. Mais il y a un truc que je veux absolument vous montrer. Cliquez sur l'onglet « response » ;

```

1 <!DOCTYPE html>
2 <html lang="fr">
3   <head>
4     <meta charset="UTF-8">
5     <title>Mon 4ème Helloworld</title>
6     <link rel="stylesheet" href="helloworld.css" type="text/css" />
7   </head>
8   <body>
9     <div class="annonce">
10       Bonjour le monde
11       <br>
12       <span style="font-size:28px">
13         jouons avec les variables
14       </span>
15     </div>
16     <script src="helloworld.js"></script>
17   </body>
18 </html>
19

```

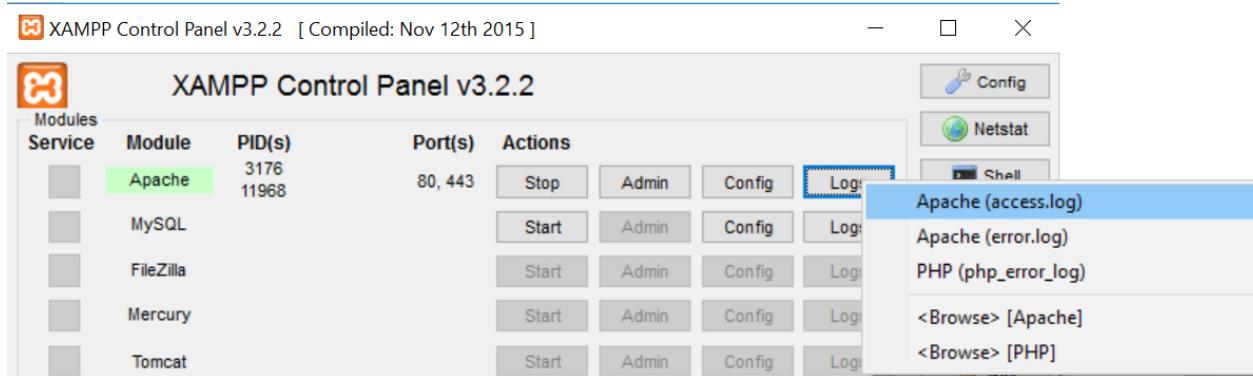
Vous le reconnaissiez ? C'est le code HTML produit par le script PHP. Tout est là. C'est ce bloc de

code HTML que le navigateur va exploiter pour construire la page. C'est aussi en analysant ce code que le navigateur va comprendre qu'il doit lancer des requêtes HTTP pour demander la livraison de ressources complémentaires (les fameux fichiers « helloworld.js » et « helloworld.css »).

Voilà, vous savez tout ou presque, sur ce qui se passe côté navigateur (côté « client » comme on le dit souvent).

Mais que se passe-t-il dans le même temps côté serveur, donc en particulier du côté d'Apache. Eh bien, nous pouvons savoir beaucoup de choses en analysant les fichiers de log générés par Apache.

Nous allons nous intéresser en particulier au fichier « access.log ». Sur Xampp, nous pouvons y accéder en cliquant sur le bouton « Logs » :



Nous allons prélever quelques échantillons de ce fichier :

- le 10 juillet à 10h42, requête vers la ressource « helloworldx.php » qui n'existe pas (on retrouve le fameux code erreur 404)

```
127.0.0.1 - - [10/Jul/2017:10:21:49 +0200] "GET /essaiphp/helloworldx.php HTTP/1.1" 404 1149 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0) Gecko/20100101 Firefox/54.0"
```

- Un peu plus tard le même jour, nouvelle requête vers la ressource « helloworld4.php » (avec code retour 200), suivi de 2 requêtes complémentaires (pour les fichiers CSS et JS) :

```
127.0.0.1 - - [10/Jul/2017:10:42:10 +0200] "GET /essaiphp/helloworld4.php HTTP/1.1" 200 443 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0) Gecko/20100101 Firefox/54.0"
::1 - - [10/Jul/2017:10:42:41 +0200] "GET /essaiphp/helloworld4.php HTTP/1.1" 200 443 "-" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
::1 - - [10/Jul/2017:10:42:41 +0200] "GET /essaiphp/helloworld.css HTTP/1.1" 304 - "http://localhost/essaiphp/helloworld4.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
::1 - - [10/Jul/2017:10:42:41 +0200] "GET /essaiphp/helloworld.js HTTP/1.1" 304 - "http://localhost/essaiphp/helloworld4.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
```

Concernant la première ressource, le fameux fichier « helloworld4.php », Apache a détecté grâce à l'extension du fichier, qu'il s'agissait d'un script PHP. Apache a vérifié que cette ressource était bien présente sur le serveur. S'il ne l'avait pas trouvée, il aurait renvoyé un code retour 404, mais comme la ressource est bien présente, Apache a demandé à l'interpréteur PHP de l'exécuter.

L'interpréteur PHP a exécuté le script et renvoyé le bloc de code HTML produit par le script PHP. Apache a pris ce bloc de code, en a fait un joli paquet cadeau avec un ruban portant la mention « réponse HTTP », et l'a envoyé au client (notre navigateur). C'est cette fameuse réponse que nous regardions en détail, si vous prenez la peine de remonter de 2 pages en arrière.

Voilà, vous savez tout, ou presque, sur la manière dont fonctionne un échange de type « client-serveur » basé sur le protocole HTTP. Nous explorerons quelques aspects complémentaires (que j'ai volontairement laissés de côté) quand nous aborderons la gestion des formulaires.

Tout ça peut sembler un peu compliqué. J'espère avoir néanmoins réussi à vous présenter la chose de manière pas trop barbante.

C'est très intéressant de connaître les mécanismes que nous venons de décrire. C'est bien sûr intéressant d'un point de vue intellectuel, mais c'est surtout très utile quand vous commencez à rencontrer des difficultés, et que vous ne parvenez pas à comprendre d'où vient le problème. Est-ce que la requête HTTP est mal formatée ? Il se peut tout simplement que l'URL soit fausse, et que la requête ne parvienne pas au serveur. Ou alors la requête est correcte, mais il y a un problème côté serveur. Grâce aux connaissances que vous venez d'acquérir, vous arriverez à comprendre ce qui se passe, et vous ne serez jamais pris au dépourvu.

3.9 Déconstruire un « hello world »

Nous avons vu avec le 4^{ème} Hello world, que le code commençait à devenir confus. Je vous le remets ci-dessous, pour vous rafraîchir la mémoire :

```
<?php
    $titre = 'Mon 4ème Helloworld';
    $encodage = 'UTF-8';
    $message1 = "Bonjour le monde";
    $message2 = "jouons avec les variables";
    $taille_message2 = 28;
?><!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="<?php echo $encodage; ?>">
        <title><?php echo $titre; ?></title>
        <link rel="stylesheet" href="helloworld.css" media="screen">
    </head>
    <body>
        <div class="annonce">
            <?php
                // bon, il paraît que je dois mettre mon code ici
                echo $message1 . PHP_EOL ;
                echo '<br>' . PHP_EOL ;
            ?>
            <span style="font-size:<?php echo $taille_message2; ?>px">
                <?php
                    echo $message2 . PHP_EOL ;
                ?>
            </span>
        </div>
        <script src="helloworld.js"></script>
    </body>
</html>
```

... Non, franchement, ce code ne fait que 29 lignes, et on se croirait déjà dans l'auberge espagnole 😊. Comment pourrait s'y prendre pour que ce code soit moins confus ?

Nous pourrions commencer par circonscrire le code HTML dans des variables.

Regardons ce que cela pourrait donner. Voici le script « helloworld4a.php » :

```

<?php
$titre = 'Mon 4ème Helloworld revisité';
$encodage = 'UTF-8';
$message1 = "Bonjour le monde";
$message2 = "jouons avec les variables";
$taille_message2 = 28;

$doctype = '<!DOCTYPE html>' . PHP_EOL;
$page_debut = '<html lang="fr">' . PHP_EOL;
$page_fin = '</html>';
$entete_debut = '<head>' . PHP_EOL ;
$entete_fin = '</head>' . PHP_EOL;
$entete_bloc = '<meta charset="' . $encodage . '">' . PHP_EOL .
    '<title>' . $titre . '</title>' . PHP_EOL .
    '<link rel="stylesheet" href="helloworld.css" media="screen">' . PHP_EOL ;

$corps_entete = '<body>' . PHP_EOL ;
$corps_bloc = '<div class="annonce">' . PHP_EOL .
    $message1 . PHP_EOL .
    '<br>' . PHP_EOL .
    '<span style = "font-size:' . $taille_message2. 'px" >' . $message2 .
    PHP_EOL . '</span >' . PHP_EOL;

$corps_bloc .= '</div >' . PHP_EOL . '<script src="helloworld.js" ></script
>' . PHP_EOL ;
$corps_fin = '</body >' ;

// Génération du code HTML avec les "echo" ci-dessous
echo $doctype;
echo $page_debut;
echo $entete_debut;
echo $entete_bloc;
echo $entete_fin;
echo $corps_entete;
echo $corps_bloc;
echo $corps_fin;
echo $page_fin;

```

Bon, quelques explications s'imposent :

- Premier détail, vous remarquez qu'il n'y a plus qu'une seule balise d'ouverture du code PHP, celle qui se trouve au tout début : **<?php**
- Du coup on ne passe plus son temps à ouvrir et fermer les balises PHP comme on le faisait dans la version antérieure
- Dans cette nouvelle version, j'ai créé tout un jeu de variables de type « chaînes de caractères », variables que j'ai alimentées avec des bouts de code HTML (une pour le

- doctype, une pour l'entête de page, etc...)
- Pour générer les différents blocs de code HTML, j'ai usé, et même abusé, de la concaténation PHP, dont je rappelle qu'elle se fait en PHP avec le point (.)
 - Pour produire un code HTML lisible, j'ai aussi utilisé la constante PHP_EOL, dont je rappelle qu'elle permet de générer un saut de ligne à l'intérieur du code HTML.

Si vous oubliez d'insérer des sauts de ligne à l'intérieur de votre code HTML, ce n'est pas dramatique, cela ne change rien pour le navigateur et pour le rendu final de la page. En revanche, un code HTML sans saut de ligne est plus difficile à lire, et il est du même coup plus difficile de repérer des erreurs s'il y en a.

Voici un exemple de code source HTML produit par une version du script PHP n'utilisant pas la constante PHP_EOL (donc pas de saut de ligne) :

```
<!DOCTYPE html><html lang="fr"><head><meta charset="UTF-8"><title>Mon 4ème Helloworld revisité</title><link rel="stylesheet" href="helloworld.css" media="screen"></head><body><div class="annonce">Bonjour le monde<br><span style = "font-size:28px" > jouons avec les variables</span></div><script src="helloworld.js" ></script ></body > </html>
```

Argh !! c'est imbuvable !!

Voici le même code HTML avec des sauts de ligne :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<title>Mon 4ème Helloworld</title>
<link rel="stylesheet" href="helloworld.css" media="screen">
</head>
<body>
<div class="annonce">
Bonjour le monde
<br>
<span style = "font-size:28px" > jouons avec les variables
</span >
</div >
<script src="helloworld.js" ></script >
</body > </html>
```

Ah, là on respire mieux 😊.

Tiens, pendant que vous saisissez votre code, peut être avez-vous rencontré quelques erreurs. Je vais partir du principe que vous êtes très fort, et que vous n'avez commis aucune erreur. Aussi, comme je souhaite vous montrer quelques types d'erreur que vous êtes susceptible de rencontrer, je vais modifier la fin de mon script PHP en introduisant 2 erreurs que j'ai indiquées avec des commentaires :

```
// Génération du code HTML avec les "echo" ci-dessous
echo $doctype;
echo $page_debut;
echo $entete_debut;
echo entete_bloc; // aïe, j'ai oublié le dollar devant le nom de la variable
echo $entetex_fin; // aïe, le nom de la variable est erroné, elle n'existe pas
echo $corps_entete;
echo $corps_bloc;
echo $corps_fin;
echo $page_fin;
```

Voyons ce que ça donne dans le navigateur :

Notice: Use of undefined constant entete_bloc - assumed 'entete_bloc'
in D:\Tools\xampp\htdocs\essaiphp\helloworld4ax.php on line 30
entete_bloc

Notice: Undefined variable: entetex_fin
in D:\Tools\xampp\htdocs\essaiphp\helloworld4ax.php on line 31

Il s'agit de messages de type « notice », donc des alertes. Nous avons ici 2 alertes, produites par le code se trouvant sur les lignes 30 et 31, que j'ai modifiées il y a quelques instants. A noter que l'on retrouve ces alertes dans le fichier « php_error.log », que vous pouvez consulter dans Xampp en passant par ce menu :



Bon, c'est quoi une alerte de type « notice » ? Il s'agit d'une anomalie potentielle ou avérée (l'interpréteur PHP ne prend pas parti, c'est à vous d'apprecier la gravité du problème).

Les alertes de type « notice » apparaissent car la configuration de PHP dans mon stack PHP est faite de telle manière que ces alertes sont envoyées directement dans le flux de données destiné au navigateur.

En revanche, sur un environnement de production, l'affichage de ces erreurs est bloqué pour ne pas perturber l'utilisateur. Mieux vaut s'intéresser au plus tôt à ces erreurs, et les éliminer dès qu'elles apparaissent.

J'ai trouvé dans la doc officielle de PHP cet extrait que je trouve intéressant :

Activer le rapport d'erreur de niveau E_NOTICE durant le développement a des avantages. En terme de débogage, les messages d'alertes vous signalent des bogues potentiels dans votre code. Par exemple, l'utilisation de valeurs non initialisées est signalée. Il est aussi plus pratique pour trouver des coquilles, et, ainsi, gagner du temps. Les messages NOTICE vous signaleront aussi les mauvaises pratiques de codage. Par exemple \$arr[item] doit toujours être écrit \$arr['item'] car PHP va considérer "item" comme une constante, au premier abord. Si cette constante n'est pas définie, alors il va l'utiliser comme une chaîne.

Source : <http://php.net/manual/fr/errorfunc.configuration.php#ini.error-reporting>

Bon, mais franchement, vous en pensez quoi de ce code ? Est-il vraiment plus lisible que le précédent ? N'y aurait-il pas mieux à faire ? Si bien sûr, en fait il y a plein de manières différentes de faire, mais pour nous devons au préalable en savoir un peu plus sur le fonctionnement des chaînes de caractères en PHP. C'est ce que je vous propose d'aborder dans le chapitre suivant (et nous en profiterons pour voir une dernière version de « hello world »).

4 Les bases du langage PHP

4.1 Les chaînes de caractères

Il existe 4 manières de créer des chaînes de caractères en PHP.

1. Chaîne délimitée par des apostrophes
2. Chaîne délimitée par des guillemets
3. Syntaxe Heredoc
4. Syntaxe Nowdoc (apparue sur PHP 5.3)

Entre les techniques 1 et 2, il y a une différence importante, que je me propose d'expliquer en prenant un petit bout de notre dernier « hello world ».

`jouons avec les variables`

Pour créer cette « span » avec des apostrophes, on peut s'y prendre de plusieurs manières.

On peut par exemple initialiser une variable, puis la compléter par petits bouts, en utilisant le raccourci « .= » qui signifie grosso modo « je te colle au derrière le bout de code qui se trouve à droite du égal » :

```
$mon_message = '<span style="font-size:' ;
$mon_message .= $taille_message2 ;
$mon_message .= 'px">';
$mon_message .= $message2;
$mon_message .= '</span>' ;
```

Mais on peut aussi faire du « tout en un » :

```
$mon_message = '<span style="font-size:' .
    $taille_message2.'px">'.$message2.'</span>' ;
```

Les chaînes délimitées avec des guillemets ont une particularité que celles avec apostrophes n'ont pas : on peut y insérer des variables, ce qui va déclencher un mécanisme d'interpolation ayant pour effet de remplacer les variables par leur contenu.

Exemple :

```
$nom1 = 'titi';
$nom2 = 'gros minet';
echo "$nom1 a piégé $nom2"; //=> titi a piégé gros minet
echo '$nom1 a piégé $nom2'; //=> $nom1 a piégé $nom2
```

Si on reprend notre exemple de la balise « span », cela donne :

```
$mon_message = "<span style=\"$taille_message2px\">
$message2</span>" ;
```

On peut aussi délimiter les variables interpolées en utilisant des accolades (personnellement j'aime bien, mais ce n'est pas obligatoire) :

```
$mon_message = "<span style=\"$font-size:{$taille_message2}px\">
$message2</span>" ;
```

A l'arrivée, on va bien obtenir ceci :

`jouons avec les variables`

Mais au fait, pourquoi ai-je ajouté des barres obliques qu'on appelle « antislash » ? Il s'agit d'un caractère qu'on appelle « caractère d'échappement ». Il nous sert à indiquer que le caractère qui le suit - en l'occurrence il s'agit d'un guillemet - ne doit pas être interprété par l'interpréteur PHP. S'il était interprété, il serait considéré à tort comme caractère de fermeture de la chaîne qui a été « ouverte » avec ce même caractère guillemet.

Nous avons vu les chaînes délimitées par des guillemets et des apostrophes.

Bon, et la syntaxe HEREDOC, c'est quoi au juste ?

Le plus simple c'est de l'étudier au travers d'un exemple. La syntaxe HEREDOC se reconnaît facilement car elle début avec ce symbole <<<, suivi du nom du bloc (et là vous pouvez vous lâcher, c'est « open-bar ») :

```
$varbidon = 'je veux';

$mon_texte = <<<BLOC_TEXTE
j'écris ce que je veux ici, c'est génial,
pas besoin de s'embêter avec des "caractères d'échappement"
et je peux placer des variables interpolées où $varbidon
BLOC_TEXTE;
```

La syntaxe HEREDOC, honnêtement j'aime beaucoup ☺. Il y a juste une petite contrainte, c'est que le nom du bloc que j'ai appelé ici « BLOC_TEXTE » doit impérativement être écrit – sur la ligne délimitant la fin du bloc – à partir du premier caractère de la ligne. Et il ne faut mettre aucun caractère (même pas un blanc), après le point virgule de cette même ligne. A part ça ce petit détail, c'est impeccable.

La syntaxe NOWDOC, apparue avec PHP 5.3, pour l'instant je vous avoue que je ne lui ai pas trouvé beaucoup d'utilité. En résumé, c'est la même chose que HEREDOC, mais sans l'interpolation de variable.

Fort de toutes ces connaissances, nous pouvons réécrire notre « hello world » en exploitant la technique d'interpolation et la syntaxe HEREDOC. Voici une possibilité d'implémentation :

```
<?php
$titre = 'Mon 4ème Helloworld';
$encodage = 'UTF-8';
$message1 = "Bonjour le monde";
$message2 = "jouons avec les variables";
$taille_message2 = 28;

$mon_message = "<span style=\"font-size:{$taille_message2}px\>{$message2}</span>" ;

$mon_html = <<<BLOC_HTML
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset=\"$encodage\">
        <title>$titre</title>
        <link rel="stylesheet" href="helloworld.css" media="screen">
    </head>
    <body>
        <div class="annonce">
            $message1
            <br>
            $mon_message
        </div>
        <script src="helloworld.js"></script>
    </body>
</html>
BLOC_HTML;

echo $mon_html;
```

Si vous souhaitez approfondir vos connaissances sur les chaînes de caractères PHP :
<http://php.net/manual/fr/language.types.string.php>

4.2 Introduction aux variables, constantes, conditions, boucles

Dans les chapitres précédents, et en particulier dans certains exemples de « hello world », nous avons utilisé quelques variables et quelques constantes. Nous allons dans ce nouveau chapitre approfondir ces notions, et quelques autres.

Pour tester les différents exemples présentés dans ce chapitre, vous pouvez détourner un des « hello world » que nous avons créé dans les chapitres précédents, ou vous pouvez effectuer vos tests directement sur le site « PHP Sandbox » dont je rappelle le lien ci-dessous :

<http://sandbox.onlinephpfunctions.com/>

4.2.1 Variables, types, conditions

En PHP, comme sur la plupart des langages de programmation, vous avez besoin de conserver temporairement des informations, en vue de pouvoir les utiliser à différentes étapes du script durant son exécution. Imaginez les variables comme les tiroirs d'une grande armoire, cette grande armoire étant la mémoire de votre ordinateur (ou plus exactement de la mémoire allouée à l'interpréteur PHP par votre système).

Une variable en PHP se définit par un nom précédé du symbole dollar (\$). Nous en avons déjà vu quelques exemples, tels que :

```
$titre = 'Mon 4ème Helloworld';
$encodage = 'UTF-8';
$message1 = "Bonjour le monde";
$message2 = "jouons avec les variables";
$taille_message2 = 28;
```

PHP fonctionne selon le principe du typage faible, notamment comme Javascript. Cela signifie que le type d'une variable dépend de son contenu. Si ce contenu change, le type de la variable change aussi, et ces changements peuvent intervenir au sein d'un même script :

```
$taille_message2 = '28'; // variable de type string (chaîne)
$taille_message2 = 28; // variable de type integer (nombre entier)
```

Dans des langages utilisant un typage fort, comme Java, le type de la variable est précisé dès sa création, et on n'a pas le droit de le modifier en cours de route.

En plus des variables personnalisées (celle que vous créez vous-même), PHP fournit un certain nombre de variables prédéfinies. Comme pour les variables personnalisées, le nom des variables prédéfinies commence par un dollar. Mais le nom lui-même est en majuscule. On trouve par exemple `$_POST`, `$_GET`, `$_SESSION`, `$GLOBALS`, etc... Vous trouverez un descriptif détaillé sur le lien suivant, mais n'y passez pas trop de temps, nous en étudierons un certain nombre au fil de l'eau :

<http://php.net/manual/fr/reserved.variables.php>

Je rappelle donc que PHP fonctionne sur le principe du typage faible, que les anglophones désignent par « weakly typed » (ou « loosely typed »). Du coup, il est parfois – et même souvent – nécessaire de connaître le type d'une variable, pour déterminer de quelle manière on va pouvoir la manipuler (si c'est une chaîne, on n'emploiera pas les mêmes fonctions que s'il s'agit d'un nombre, ou on pourra être amené à effectuer une conversion de type, en préalable à un traitement).

Heureusement, on dispose d'un certain nombre de fonctions bien pratiques, permettant de connaître et manipuler le type et le contenu des variables. Parmi ces fonctions, on trouve : `gettype()`, `settype()`, `isset()`, `unset()`, `empty()`, plus un jeu de fonctions telles que `is_int()`, `is_array()`, `is_numeric()`, etc...

Avant de présenter certaines de ces fonctions, il me semble utile de préciser que plusieurs de ces fonctions renvoient une valeur booléenne, qui peut être « vrai » (en anglais : « true »), ou « faux » (en anglais : « false »). Ce résultat « vrai » ou « faux » peut être stocké dans une variable, qui va du coup être de type « booléen ». Tout cela se raccroche bien évidemment à l'algèbre booléenne, qui tire son nom d'un célèbre logicien, mathématicien et philosophe britannique, j'ai nommé Georges Boole : https://fr.wikipedia.org/wiki/George_Boole

Certains langages de programmation avec lesquels on travaillait dans les années 80 et 90 ne savaient pas gérer le type booléen. Du coup, chaque fois que l'on avait besoin de stocker le résultat d'un test, on utilisait traditionnellement une variable numérique de type « integer » (nombre entier) dans laquelle on stockait 0 (zéro) pour la valeur « faux » et 1 pour la valeur « vrai ». Si je vous en parle ici, c'est pour vous montrer qu'il reste un peu de cette logique dans le PHP. Si j'affiche une variable booléenne contenant « true », je ne vais pas obtenir « vrai », mais « 1 ». Et pour « false », c'est encore pire, car j'obtiens un blanc (même pas un zéro). Pour clarifier mon propos, j'ai mis un petit de code PHP en exemple sur la page suivante. Je vous invite à tester ces bouts de code sur « PHP Sandbox » ou dans votre stack PHP local.

```

$test = true;
echo $test . PHP_EOL; // affiche 1
if ($test) {
    echo "vrai".PHP_EOL;
} else {
    echo "ce texte ne devrait pas s'afficher";
}

$test = false;
echo $test . PHP_EOL; // affiche un blanc, et non pas un 0
if (!$test) {
    echo "faux".PHP_EOL;
} else {
    echo "ce texte ne devrait pas s'afficher";
}

```

On voit que les valeurs booléennes « true » et « false » ne doivent en aucun cas être utilisées directement pour de l'affichage. Il faut toujours passer par un test comme ceux que vous pouvez lire ci-contre.

Tiens, en parlant de tests, vous allez revoir cette structure que l'on appelle « test » à plusieurs reprises, alors autant en parler un petit peu maintenant.

Quand je souhaite déterminer si une condition est remplie (par exemple si 1 est bien égal à 1), je peux écrire ceci :

```

if (1==1) {
    echo "vrai".PHP_EOL;
} else {
    echo "faux".PHP_EOL; // ne se produira jamais compte tenu de la condition
}

```

En notation algorithmique on écrirait ceci :

```

Si 1==1
    Alors
        Afficher "vrai" suivi d'un saut de ligne
    Sinon
        Afficher "faux" suivi d'un saut de ligne
FinSi

```

Le test ci-dessus est carrément idiot, c'était juste pour vous montrer le principe. Dans la « vraie vie » vous serez parfois confronté au besoin d'effectuer plusieurs fois un même test. Pour éviter une répétition pénible, qui peut être source d'erreur, vous aurez intérêt à stocker le résultat de ce test dans une variable. Vous pourrez ainsi évaluer le contenu de cette variable (pour voir s'il est à « vrai » ou « faux»), au moment où cela vous arrange. Dans le cas de notre test précédent, qui est toujours aussi idiot, cela donnerait cela :

```
$test = 1==1; // stocke la valeur "true" dans la variable $test
if ($test == true) {
    echo "vrai".PHP_EOL;
} else {
    echo "faux".PHP_EOL;
}
```

PHP nous permet de simplifier l'écriture ci-dessus, en écrivant la condition ci-dessous, qui est strictement équivalente :

```
if ($test) {
    echo "vrai".PHP_EOL;
} else {
    echo "faux".PHP_EOL;
}
```

Dans un test, nous pouvons souhaiter privilégier le cas où la condition renvoie « faux », ou en tout cas nous pouvons

```
if ($test == false) {
    echo "false".PHP_EOL;
} else {
    echo "vrai".PHP_EOL;
}
```

PHP nous permet - là encore - de simplifier l'écriture ci-dessus, en écrivant la condition ci-dessous (avec le point d'exclamation, qui peut se lire « non vrai ») :

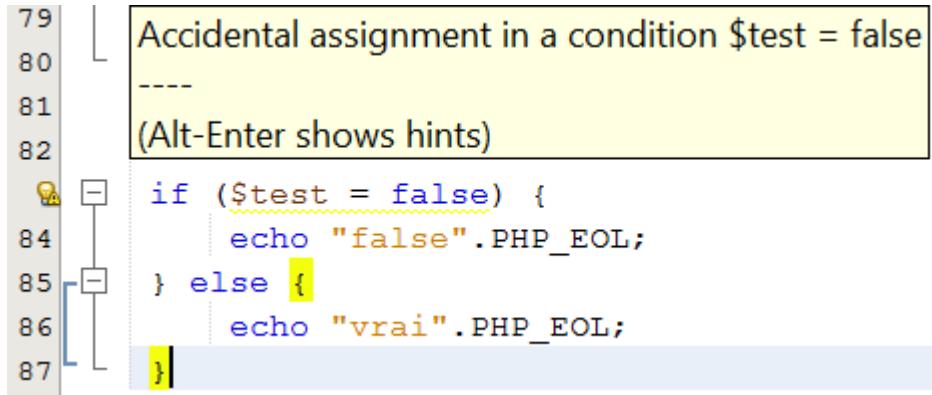
```
if ( !$test) {
    echo "false".PHP_EOL;
} else {
    echo "vrai".PHP_EOL;
}
```

Vous avez sans doute remarqué que nous avons effectué le test d'égalité avec un double égal (==). En effet, le simple égal (=) correspond à une instruction d'affectation. Si dans un moment d'inattention, vous écrivez ceci :

```
if ($test = false) {
    echo "false".PHP_EOL;
} else {
    echo "vrai".PHP_EOL;
}
```

... en réalité vous venez d'écrire : j'affecte la valeur « false » à la variable \$test, et comme cette affectation s'est bien passée (il n'y a d'ailleurs aucune raison pour qu'elle se passe mal), cette opération d'affectation renvoie implicitement la valeur « true » (donc « vrai ») : donc notre test est faussé. Je vous garantis que ce genre de bêtise arrive souvent, et même à des développeurs chevronnés.

Certains IDE fournissent une assistance qui permet de se prémunir contre ce genre d'erreur. Par exemple, Netbeans fait ça plutôt bien. Dans l'exemple ci-dessous, j'ai placé le pointeur de la souris sur le picto , qui se trouve au début de la ligne du « if », et un message est venu m'avertir du fait que j'avais probablement écrit une ânerie :



```

79
80
81
82
83
84
85
86
87

```

```

if ($test = false) {
    echo "false".PHP_EOL;
} else {
    echo "vrai".PHP_EOL;
}

```

Dans l'annexe 5.1, vous trouverez quelques approfondissements autour des opérateurs PHP. Je vous laisse le soin d'étudier ce chapitre à tête reposée.

Je vous propose maintenant d'étudier quelques fonctions PHP qui sont bien pratiques pour connaître et manipuler le type et le contenu des variables. Nous les avons listées quelques pages plus haut, mais je les rappelle ici : [gettype\(\)](#), [settype\(\)](#), [isset\(\)](#), [unset\(\)](#), [empty\(\)](#), plus un jeu de fonctions telles que [is_int\(\)](#), [is_array\(\)](#), [is_numeric\(\)](#), etc...

gettype()

`gettype()` détermine le type d'une variable reçue en paramètre, et renvoie une chaîne qui peut prendre l'une des valeurs suivante : integer, double, string, array, object, boolean, resource, NULL et unknown.

```

$mavar = '123';
if(gettype($mavar) == "string") {
    echo ($mavar);
}

```

settype()

`settype()` définit explicitement le type d'une variable (transmise en premier paramètre), par rapport à un type transmis sous forme de chaîne (en second paramètre). Les valeurs possibles sont : integer, double, string, array, object. Si le type est correctement défini, alors `SetType()` renvoie true, sinon la fonction retourne faux.

```
$mavar = '123';
if(settype($mavar, "integer")) {
    echo("Cette variable est bien de type entier");
}
```

isset()

`isset()` est utilisée pour déterminer si une variable est définie (renvoie "true") ou pas (renvoie "false").

```
$mavar = '123';
if( isset($mavar) ) {
    echo("Cette variable a bien une valeur définie");
}
```

On peut aussi se servir de la fonction `isset()` pour tester la présence d'un poste dans un tableau, mais nous en reparlerons dans le chapitre dédié aux tableaux.

unset()

`unset()` est utilisée pour détruire une variable transmise en paramètre (et ça fonctionne aussi pour supprimer un poste de tableau).

```
unset($mavar);
if(isset($mavar)) {
    echo("Ce texte ne s'imprimera jamais !");
}
```

empty()

`empty()` est l'exact opposé de la fonction `isset()`. Elle retourne « vrai » si la variable n'est pas définie, et « faux » si elle est définie. Dans le cas de la fonction « `isset()` » de l'exemple précédent, on pourrait obtenir strictement le même résultat en écrivant ceci :

```
unset($mavar);
if( !empty($mavar) ) {
    echo("Ce texte ne s'imprimera jamais !");
}
```

is_[datatype] ()

Les fonctions `is_int()`, `is_integer()`, et `is_long()` vérifient si la variable transmise en paramètre est de type "entier".

Le type de données "décimal" dispose également de trois fonctions : `is_double()`, `is_float()`, et `is_real()`. La fonction `is_string()` indique si la variable est une chaîne.

Les fonctions `is_array()` et `is_object()` fonctionnent sur le même principe, avec leurs types de données respectifs.

```
$somevar = "ceci est une chaîne";
if(is_string($somevar)) {
    echo("Cette variable est une chaîne");
}
```

Pour vous aider à y voir clair :

- Si vous souhaitez vous assurer qu'une variable est bien un nombre, utilisez la fonction `is_numeric()` pour vérifier sa valeur.
- Vous pouvez forcer une variable à « adopter » le type « nombre entier » en utilisant la fonction `intval()`, ou en plaçant `(int)` devant la variable (cf. exercice ci-après).
- Si la valeur doit être un tableau, utilisez `is_array()`.
- Pour tester si une variable est une chaîne, utilisez `is_string()`. Pour la forcer à devenir une chaîne, utiliser `strval()`.
- Si la valeur devrait être nulle, utiliser `is_null()`.
- Si la valeur doit être définie, utilisez `isset()`.

Je vous propose un petit exercice : lisez le code ci-dessous, essayez de déterminer lesquels des messages 1, 2, 3 et 4 vont s'afficher, puis testez le :

```
$var1 = '123a';
$var2 = 123;
if( $var1 == $var2){
    echo("Ces variables contiennent la même valeur (1)".PHP_EOL);
}
if( $var1 === $var2){
    echo("Ces variables contiennent la même valeur (2)".PHP_EOL);
}
if( (int)$var1 == (int)$var2{
    echo("Ces variables contiennent la même valeur (3)".PHP_EOL);
}
if( (int)$var1 === (int)$var2{
    echo("Ces variables contiennent la même valeur (4)".PHP_EOL);
}
```

Si vous avez du mal à comprendre ce qui se passe, testez le code suivant :

```
echo (int)$var1;
```

Dans ce chapitre d'introduction aux variables, nous avons présenté quelques types de données très simples. Dans un prochain chapitre, nous étudierons certains types particuliers comme les tableaux. Mais avant cela, je vous propose de faire d'abord un petit détour par les constantes.

Ah zut, j'ai failli conclure ce chapitre sans parler de l'instruction « switch ». C'eut été dommage, car elle est très pratique pour l'écriture de certains types de tests (et nous l'utiliserons de temps à autre dans ce cours). Voici un exemple emprunté à php.net (et légèrement remanié), dans lequel un exemple de test à base de « if » et un exemple à base de « switch » sont présentés en parallèle. Ils sont strictement équivalents :

```
$i = isset($_GET['test'])?(int) $_GET['test']:0;

if ($i == 0) {
    echo "i égal 0";
} elseif ($i == 1) {
    echo "i égal 1";
} elseif ($i == 2) {
    echo "i égal 2";
} else {
    echo 'je ne sers à rien, je suis là pour l\'exemple';
}

switch ($i) {
    case 0: {
        echo "i égal 0";
        break;
    }
    case 1: {
        echo "i égal 1";
        break;
    }
    case 2: {
        echo "i égal 2";
        break;
    }
    default: {
        echo 'je ne sers à rien, je suis là pour l\'exemple';
    }
}
```

Je pense que cette technique ne présente pas de difficulté particulière, je vous recommande de lire la documentation de php.net pour quelques exemples complémentaires :
<http://php.net/manual/fr/control-structures.switch.php>

A noter que les accolades après chaque « case » sont facultatives. Mais personnellement, je préfère les mettre, je trouve que leur présence contribue à la lisibilité du code. Et dans la « vraie vie » il est rare que l'on ait qu'une seule instruction à exécuter à l'intérieur d'un « case », on en a en général plusieurs, et dans ce cas les accolades sont indispensables.

4.2.2 Constantes

Les constantes fonctionnent elles-aussi comme les tiroirs de la même grande armoire (relative à la mémoire allouée à PHP par le système). Mais des tiroirs dont le contenu, une fois défini, n'est pas modifiable.

La déclaration d'une constante en PHP se fait au moyen de la fonction define() :

```
define('MA_DATA1', 23);
define("MA_DATA2", 'hello!!!!');
echo MA_DATA1;
echo MA_DATA2;
```

On voit que les constantes n'utilisent pas le caractère dollar.

Par convention on déclare les constantes avec un nom en majuscule. Cela permet de les repérer plus facilement à l'intérieur du code.

Pour approfondir la notion de constante PHP :

<http://php.net/manual/fr/function.define.php>

En plus des constantes personnalisées, PHP fournit un nombre assez important de constantes standard. Vous connaissez déjà PHP_EOL, vous aurez peut être besoin de temps à autre d'utiliser PHP_VERSION, pour connaître la version courante de PHP et ainsi savoir si vous pouvez utiliser une instruction PHP standard, ou si vous devez faire appel à une solution de substitution (si la fonction qui vous intéresse n'existe pas dans certaines versions de PHP).

Je vous invite à jeter un coup d'œil à la liste des constantes prédéfinies (on dit aussi « constantes réservées ») présentée sur cette page :

<http://php.net/manual/fr/reserved.constants.php>

4.2.3 Tableaux et boucles

Un tableau c'est une liste ou collection de données. Si l'on reprend l'idée de « grande armoire », que nous avions évoquée lors de notre introduction aux variables, un tableau ce serait un classeur à l'intérieur d'un tiroir de notre grande armoire. Ce classeur peut contenir des données très structurées (exemple : une collection de timbre), ou au contraire pas structurées du tout (exemple : toutes les bricoles qu'un enfant ramène dans ses poches à la fin d'une journée de promenade, comme des cailloux, des trombones, des craies, des bonbons, etc...).

Pour initialiser un tableau en PHP, on peut utiliser l'une des 2 syntaxes suivantes (elles sont strictement équivalentes) :

```
$tableau = [];
$tableau = array();
```

En PHP, il existe deux grands types de tableaux :

- Les tableaux simples
- Les tableaux associatifs

Commençons par les tableaux simples.

Voici un exemple de tableau contenant une liste d'APIs spécifiques à la norme HTML5 :

```
$apis_HTML5 = array('Canvas', 'SVG', 'Touch', 'WebAudio', 'WebMIDI',
    'LocalStorage', 'SessionStorage', 'WebWorker', 'WebSocket', 'File');
```

On aurait pu obtenir strictement le même résultat en écrivant ceci :

```
$apis_HTML5 = array();
$apis_HTML5[] = "Canvas";
$apis_HTML5[] = "SVG";
$apis_HTML5[] = "Touch";
$apis_HTML5[] = "WebAudio";
$apis_HTML5[] = "WebMIDI";
$apis_HTML5[] = "LocalStorage";
$apis_HTML5[] = "SessionStorage";
$apis_HTML5[] = "WebWorker";
$apis_HTML5[] = "WebSocket";
$apis_HTML5[] = "File";
```

Le fait d'utiliser les crochets vides [], juste après le nom de la variable indique à PHP que l'on souhaite ajouter un nouvel élément à la fin du tableau. C'est l'équivalent de la méthode push() en Javascript.

On aurait pu également numérotter explicitement chaque poste du tableau, mais je ne recommande pas cette méthode :

```
$apis_HTML5 = [];
$apis_HTML5[0] = "Canvas";
$apis_HTML5[1] = "SVG";
$apis_HTML5[2] = "Touch";
$apis_HTML5[3] = "WebAudio";
$apis_HTML5[4] = "WebMIDI";
$apis_HTML5[5] = "LocalStorage";
$apis_HTML5[6] = "SessionStorage";
$apis_HTML5[7] = "WebWorker";
$apis_HTML5[8] = "WebSocket";
$apis_HTML5[9] = "File";
```

PHP fournit un gros jeu de fonctions pour manipuler les tableaux, vous pouvez consulter la liste détaillée sur cette page :

<http://php.net/manual/fr/ref.array.php>

Supposons que nous souhaitons trouver quel poste de notre tableau contient l'API SVG, nous pouvons utiliser la fonction `array_search()`, comme ceci :

```
$numero = array_search('SVG', $apis_HTML5);
echo $numero; // renvoie 1 car le tableau est numéroté à partir de 0
```

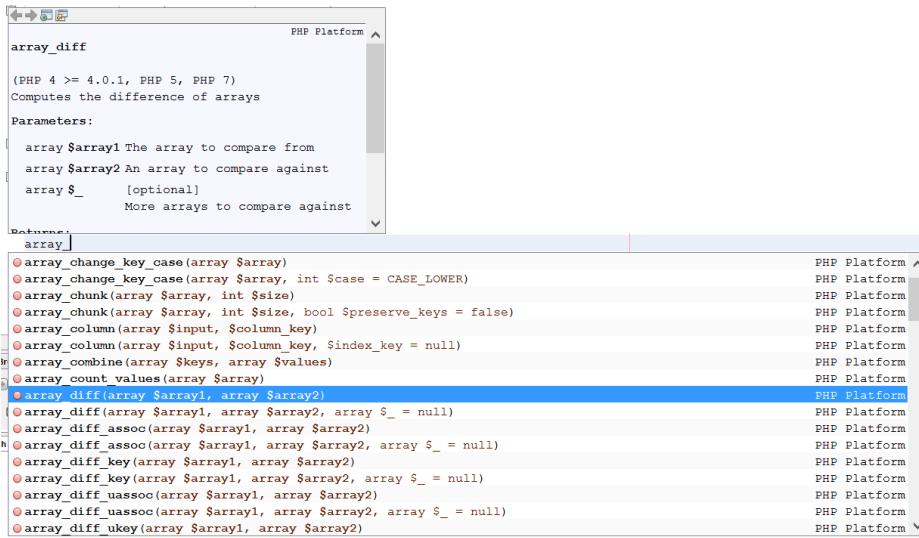
Quelques fonctions PHP intéressantes à connaître :

- `count()` : renvoie le nombre de postes d'un tableau
- `implode()` : renvoie une chaîne contenant les postes du tableau collés bout à bout, séparés par un caractère indiqué en premier paramètre de la fonction
- `explode()` : transforme une chaîne en tableau, en fonction d'un caractère séparateur...
`implode()` et `explode()` sont vraiment très pratiques
- `sort()` : tri le contenu du tableau

Attention : en PHP, il y a un manque de cohérence criant en ce qui concerne les fonctions. Beaucoup de fonctions de manipulation de tableaux ont des noms qui commencent par « `array_` », ce qui n'est pas le cas des fonctions indiquées ci-dessus, pourtant elles aussi spécialisées dans la manipulation de tableaux.

De plus, certaines fonctions reçoivent le nom du tableau en premier, second, voire troisième paramètre (peu de cohérence de ce côté-là également), et renvoient un résultat en sortie. C'est le mode de fonctionnement notamment des fonctions `count()` et `implode()`, ... tandis que d'autres fonctions ne renvoient rien en sortie et modifient le contenu du tableau qu'elle ont reçu en paramètre, c'est le cas de la fonction `sort()`.

Pour s'y retrouver dans ce joyeux bazar, heureusement que la documentation officielle sur php.net est bien faite, et que les IDE fournissent autocomplétion et aide en ligne, comme par exemple Netbeans, quand je commence à saisir « array_ » :



Parcourir la globalité d'un tableau, ce n'est pas compliqué. Pour cela vous disposez des boucles « for », « foreach » et « while » :

```
for($i=0, $ilen=count($apis_HTML5); $i<$ilen; $i++) {
    echo $i . ' => ' . $apis_HTML5[$i]. PHP_EOL;
}

foreach($apis_HTML5 as $apikey=>$apival) {
    echo $apikey . ' => ' . $apival. PHP_EOL;
}

$i = 0;
$ilen = count($apis_HTML5);
while ($i<$ilen) {
    echo $i . ' => ' . $apis_HTML5[$i]. PHP_EOL;
    $i++;
}
```

Les 3 boucles ci-dessus produisent le même résultat :

```
0 => Canvas
1 => SVG
2 => Touch
3 => WebAudio
4 => WebMIDI
5 => LocalStorage
```

```
6 => SessionStorage
7 => WebWorker
8 => WebSocket
9 => File
```

Dans le cas de la boucle « for », deux points à noter :

Il y a 3 grandes parties dans les paramètres attendus par la boucle « for », parties qui sont séparées par des points virgules :

- la partie « initialisation » qui permet d'initialiser 1 ou une plusieurs variables (dans notre exemple de la page précédente, nous initialisons simultanément les variables \$i et \$len)
- la partie « analyse de la condition de poursuite de la boucle. Dans notre exemple, nous vérifions si \$i est bien inférieure à \$len. Si c'est le cas, la boucle se poursuit, dans le cas contraire la boucle s'arrête
- la partie « incrémentation », dans laquelle on incrémente le compteur en prévision de l'itération suivante (si on l'oubliait, dans notre exemple nous partirions dans une boucle sans fin). Dans cette partie, on peut gérer le « pas » d'incrémentation, et par exemple considérer qu'on souhaite traiter un élément sur 2 ou 3.

Petite précision, écrire ceci : \$i++

... équivaut à écrire ceci : \$i = \$i + 1

... ou encore ceci : \$i+=1

On notera une différence cruciale entre les boucles « for », « foreach » et « while » :

- à l'intérieur des boucles « for » et « while », il faut impérativement utiliser l'index pour adresser un élément du tableau
`echo $i . ' => ' . $apis_HTML5[$i]. PHP_EOL;`
- à l'intérieur d'une boucle « foreach », on utilise pas directement le nom du tableau, mais le nom de la variable déclarée à droite du mot clé « as », soit ici \$apival.
- Toujours dans la boucle « foreach », vous noterez qu'on n'a pas besoin de l'index (disponible dans la variable \$apikey), pour récupérer le nom d'une API. On n'a d'ailleurs pas besoin de l'incrémenter, c'est PHP qui le fait pour nous, en récupérant l'index en cours et en le stockant dans la variable \$apikey. A noter que, si vous ne souhaitez pas

afficher ou utiliser l'index, vous pouvez simplifier l'écriture du « foreach » en écrivant ceci :

```
foreach($apis_HTML5 as $apival) {  
    echo $apival. PHP_EOL;  
}
```

Avant de passer à l'étude des tableaux associatifs, je rappelle que vous disposez des fonctions :

- `is_array()` pour vérifier si une variable est bien de type tableau.
- `count()` pour compter le nombre d'éléments d'un tableau

Ces deux fonctions sont opérationnelles aussi bien sur les tableaux simples que les tableaux associatifs.

Un tableau associatif, ce n'est pas très compliqué non plus. Voici un exemple de tableau associatif contenant un menu pas franchement équilibré :

```
$menu = array (  
    "entrée" => "terrine du chef",  
    "plat" => "entrecôte au bleu (accompagné de frites)",  
    "dessert" => "mousse au chocolat"  
) ;  
foreach($menu as $menukey=>$menuval) {  
    echo $menukey . ' : ' . $menuval. PHP_EOL;  
}
```

Résultat obtenu avec la boucle « foreach » :

```
entrée : terrine du chef  
plat : entrecôte au bleu (accompagné de frites)  
dessert : mousse au chocolat
```

Vous voyez ce n'est pas très compliqué. Il faut juste noter que les boucles « for » et « while » ne sont pas utilisables avec un tableau associatif. Seule la boucle « foreach » est véritablement adaptée à ce type de tableau.

Maintenant, supposons que nous souhaitions gérer un tableau contenant les menus de la semaine. Nous pouvons dans ce cas créer un tableau associatif de tableaux associatifs (on emploie souvent le terme de « tableaux imbriqués » pour désigner ce cas de figure).

Voici un exemple de tableau contenant des menus allant du lundi au mercredi. Chaque poste du tableau principal \$menu_sem contient lui-même 3 postes de tableau correspondant aux 3 plats principaux d'un repas :

```
$menu_sem = [];
$menu_sem['lundi']= array (
    "entrée" => "terrine du chef",
    "plat" => "entrecôte au bleu accompagné de frites",
    "dessert" => "mousse au chocolat"
);
$menu_sem['mardi']= array (
    "entrée" => "macédoine de légumes",
    "plat" => "hamburger (accompagné de frites)",
    "dessert" => "crème caramel"
);
$menu_sem['mercredi']= array (
    "entrée" => "avocat au surimi",
    "plat" => "cordon bleu accompagné de frites",
    "dessert" => "mousse au chocolat"
);
```

Voici un exemple de script PHP avec deux boucles « foreach » imbriquées permettant de générer une liste HTML :

```
echo '<ol>' . PHP_EOL;
foreach ($menu_sem as $menu_jour_key=>$menu_jour_val) {
    echo '<li>' . $menu_jour_key . PHP_EOL;
    echo '<ul>' . PHP_EOL;
    foreach($menu_jour_val as $menu_detail_key=>$menu_detail_val) {
        echo '<li>' . $menu_detail_key . ' : ' . $menu_detail_val . '</li>' . PHP_EOL;
    }
    echo '</ul>' . PHP_EOL;
    echo '</li>' . PHP_EOL;
}
echo '</ol>' . PHP_EOL;
```

Exercice : Etudiez le code ci-dessus. Essayez de bien comprendre le fonctionnement et le rôle des 2 boucles « foreach ». Essayez de vous représenter le code HTML qui va être généré et le résultat que l'on va obtenir visuellement. Ensuite seulement, vous pouvez consulter le résultat sur la page qui suit.

Résultat :

1. lundi

- entrée : terrine du chef
- plat : entrecôte au bleu accompagné de frites
- dessert : mousse au chocolat

2. mardi

- entrée : macédoine de légumes
- plat : hamburger (accompagné de frites)
- dessert : crème caramel

3. mercredi

- entrée : avocat au surimi
- plat : cordon bleu accompagné de frites
- dessert : mousse au chocolat

Jetons un coup d'œil au code HTML généré, cela fera un bon rappel par rapport au cours « HTML5 – Premiers pas ».

```
<ol>
<li>lundi
<ul>
<li>entrée : terrine du chef</li>
<li>plat : entrecôte au bleu accompagné de frites</li>
<li>dessert : mousse au chocolat</li>
</ul>
</li>
<li>mardi
<ul>
<li>entrée : macédoine de légumes</li>
<li>plat : hamburger (accompagné de frites)</li>
<li>dessert : crème caramel</li>
</ul>
</li>
<li>mercredi
<ul>
<li>entrée : avocat au surimi</li>
<li>plat : cordon bleu accompagné de frites</li>
<li>dessert : mousse au chocolat</li>
</ul>
</li>
</ol>
```

Pour conclure ce chapitre, une petite astuce à noter relative aux boucles for() et foreach(). Quelquefois vous aurez besoin d'écrire une boucle de ce genre :

```
for($i=2010;$i<2020;$i++) {  
    echo $i .PHP_EOL;  
}
```

Vous pourrez – si vous le souhaitez – remplacer cette écriture par l'écriture ci-dessous qui produira le même résultat, grâce à la fonction range() :

```
foreach(range(2000, 2020) as $i) {  
    echo $i .PHP_EOL;  
}
```

Pour une présentation détaillée de la fonction range() :

<http://php.net/manual/fr/function.range.php>

4.2.4 Commentaires

On peut insérer des commentaires dans du code PHP de plusieurs manières :

```
// ceci est un commentaire mono-ligne occupant une ligne entière  
$var1 = 123 ; // commentaire mono-ligne placé en bout de ligne  
  
/* ceci est aussi un commentaire ... */  
  
/* ... mais cette technique est généralement utilisée pour  
commenter plusieurs lignes.  
on s'en sert aussi pour mettre en commentaire  
provisoirement certaines portions de code */
```

En introduction d'une fonction, il est recommandé de placer un bloc de commentaire définissant le rôle de la fonction, ainsi que ses paramètres entrants et sortants. Les principaux IDE proposent une assistance pour la construction de ce type de commentaire.

Si l'on prend pour exemple la fonction suivante :

```
function test5($var1, $var2, $var3='', $var4=array(), $var5=0) {  
    // placer le code "métier" ici  
    return ['out1'=>$var1, 'out2'=>$var2.$var3];  
}
```

... sur la ligne précédant le mot clé « function », avec Netbeans ou PHPStorm, saisissez `/**` puis pressez la touche Entrée :

```
/**  
 * @param $var1  
 * @param $var2  
 * @param string $var3  
 * @param array $var4  
 * @param int $var5  
 * @return array  
 */
```

... vous voyez apparaître un bloc de commentaire « pré-mâché » contenant la liste des paramètres que l'IDE a été en mesure de déterminer via la ligne de déclaration de la fonction, ainsi que via le mot clé « return ». On peut dès lors compléter le bloc de commentaire en ajoutant un texte explicatif et en précisant le type des paramètres que l'IDE n'a pas été en mesure de déterminer de lui-même :

```

/**
 * Ceci est un bloc de commentaire placé avant la déclaration
 * d'une fonction pour préciser son utilité, ainsi que le
 * nombre et le type des paramètres entrants (avec @param)
 * et sortants (avec @return)
 * @param integer $var1
 * @param string $var2
 * @param string $var3
 * @param array $var4
 * @param int $var5
 * @return array
 */
function test5($var1, $var2, $var3='', $var4=array(), $var5=0) {
    // placer le code "métier" ici
    return ['out1'=>$var1, 'out2'=>$var2.$var3];
}

```

4.2.5 Fonctions

Les fonctions PHP sont assez similaires aux fonctions que l'on trouve sur la plupart des autres langages.

La création d'une fonction se fait de la façon suivante :

```

function multiplication($var1, $var2) {
    $resultat = $var1 * $var2 ;
    return $resultat ;
}

```

Explication : notre fonction `multiplication()` reçoit 2 paramètres qu'elle multiplie ensemble. Le résultat de cette multiplication est stocké dans une variable qui est ensuite envoyé en sortie de la fonction. Le stockage dans une variable n'étant pas ici indispensable, nous aurions pu simplifier le code comme ceci :

```

function multiplication($var1, $var2) {
    return $var1 * $var2 ;
}

```

PHP offre la possibilité de définir des paramètres optionnels, on les reconnaît facilement au fait qu'ils ont une valeur par défaut, comme ici pour les variables `$var3`, `$var4` et `$var5` :

```

function test($var1, $var2, $var3='', $var4=array(), $var5=0) {
    return ... ;
}

```

Si l'on souhaite qu'une fonction puisse renvoyer plusieurs valeurs en sortie, il est possible de le faire très facilement en encapsulant les différentes valeurs dans un tableau simple ou associatif. Exemple ci-dessous avec un tableau associatif :

```
function test2($var1, $var2, $var3='', $var4=array()) {  
    return ['out1'=>10, 'out2'=>$var1.$var2, 'out3'=>'titi'];  
}
```

Un autre élément important à noter, c'est la portée des variables utilisées à l'intérieur des fonctions :

- Toute variable déclarée à l'intérieur d'une fonction a une portée locale, c'est-à-dire qu'elle n'est pas visible depuis l'extérieur de la fonction
- Pour pouvoir utiliser à l'intérieur d'une fonction des variables globales, il faut les déclarer à l'intérieur de la fonction via le mot clé « global » :

```
function test3($var1, $var2, $var3='', $var4=array()) {  
    global $glo1, $glo2;  
    ...  
}
```

AVERTISSEMENT : je n'apprécie pas beaucoup les variables globales, et je sais que je ne suis pas le seul. En règle générale, j'exclus toute utilisation de variables globales au sein de fonctions, sauf cas très particulier que je prends soin de documenter dans le code, afin que les collègues qui reprendront la maintenance du code sachent précisément pourquoi j'ai fait ce choix.

Il peut être intéressant dans certains cas de recourir aux fonctions suivantes :

- [func_num_args\(\)](#) : renvoie le nombre de paramètres d'une fonction
- [func_get_args\(\)](#) : renvoie un tableau contenant les paramètres d'une fonction

Je ne rentre pas dans le détail, je vous laisse le soin de consulter la documentation officielle.

Il existe un certain nombre de fonctions PHP dédiées à la gestion de fonctions :

<http://fr.php.net/manual/fr/ref.func.php>

Apparues avec PHP 5.3, les fonctions anonymes existent aussi en Javascript (langage dans lequel elles sont très utilisées), mais elles n'ont pas connu en PHP le succès auquel on pouvait s'attendre. On commence néanmoins à les voir de plus en plus utilisées au sein de frameworks, car elles offrent beaucoup de possibilités et de souplesse.

Exemple :

```
$hello = function ($name)
{
    printf("Bonjour %s\r\n", $name);
}
$hello('le Monde'); //=> Bonjour le Monde
$hello('la planète'); //=> Bonjour la planète
```

Pour de plus amples informations sur ce sujet :

<http://php.net/manual/fr/functions.anonymous.php>

Il y a un dernier point intéressant à noter concernant les fonctions. Pour l'expliquer je vous propose d'observer le code ci-dessous :

```
/**
 * Somme de 2 termes
 * @param int $a
 * @param int $b
 * @return int
 */
function sum(int $a, int $b) {
    return $a + $b;
}

var_dump(sum(1, 2));
```

Vous remarquez que les 2 paramètres de la fonction sont précédés du mot clé « int ». Cela signifie que ces 2 paramètres ne peuvent accepter que des valeurs de type « integer ». Cela permet d'obtenir un code plus robuste. Mais cette possibilité de typer les paramètres « integer » n'est apparue qu'à partir de PHP7. Les possibilités de typage des paramètres étaient plus limitées dans les versions antérieures de PHP. Le tableau de la page suivante, emprunté au site php.net, résume bien la situation.

Type	Description	Version PHP minimum
Nom de la Classe/interface	Le paramètre doit être une instanceof de la classe ou interface donnée.	PHP 5.0.0
<code>self</code>	Le paramètre doit être une instanceof de la même classe qui a défini la méthode. Ceci ne peut être utilisé que sur les méthodes des classes et des instances.	PHP 5.0.0
array	Le paramètre doit être un array .	PHP 5.1.0
callable	Le paramètre doit être un callable valide.	PHP 5.4.0
bool	Le paramètre doit être un boolean .	PHP 7.0.0
float	Le paramètre doit être un nombre flottant (float).	PHP 7.0.0
int	Le paramètre doit être un integer .	PHP 7.0.0
string	Le paramètre doit être une string .	PHP 7.0.0

Pour de plus amples informations :

<http://php.net/manual/fr/functions.arguments.php#functions.arguments.type-declaration>

4.2.6 Include et Require

Les fonctions PHP `include()` et `require()` sont très utiles dans le processus de développement. Elles permettent d'inclure dans un script en cours d'exécution des déclarations de fonctions ou d'objets qui ont été créés dans d'autres scripts PHP.

En cas d'absence du fichier indiqué, « include » génère un warning alors que « require » génère une erreur fatale.

En règle générale, on préfère utiliser « require », car l'erreur fatale déclenchée par cette instruction nous permettra d'identifier plus rapidement une situation anormale. Or, l'absence du fichier spécifié est généralement une situation anormale (sauf cas particulier qu'il conviendra de bien documenter dans le code).

Exemple :

```
// remplace la ligne "require" par le contenu du fichier transmis en paramètre
require ("copyright.php");

// identique à "require", mais l'instruction est ignorée si le fichier
// considéré a déjà été chargé par un autre script
require_once ("header.php");

// identique à "require" mais déclenche seulement un warning en cas d'absence
// du fichier indiqué
include ("footer.php");

// identique à "include" mais ignore l'instruction si le fichier a déjà été
// chargé par un autre script
include_once ("title.php");
```

On notera qu'il est possible de « remonter » dans l'arborescence d'un site pour aller chercher un script qui se trouve dans une autre branche de l'arborescence du projet. Dans l'exemple qui suit, on remonte de 2 répertoires par rapport répertoire courant, pour aller chercher le fichier « `copyright.php` » qui se trouve dans le sous-répertoire « `lib` » :

```
require ("../../lib/copyright.php");
```

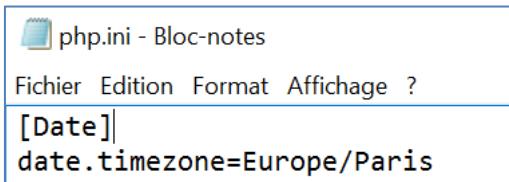
TODO : compléter ce chapitre en développant les notions de chemin d'accès et de require « automatique ».

4.3 Compléments sur PHP

4.3.1 Dates

Pour pouvoir manipuler correctement des dates en PHP, vous devez vous assurer que le fuseau horaire de référence utilisé par votre interpréteur est correctement paramétré. Le fuseau horaire, c'est la « timezone » dans le jargon de PHP. Elle est paramétrée dans votre fichier php.ini.

Si vous êtes comme moi quelque part en France, alors votre fuseau horaire de référence, c'est « Europe/Paris ».



Il y a des cas où l'on n'est pas en mesure de modifier le fuseau horaire dans le fichier « php.ini » :

- vous pourriez vous trouver dans le cas où votre application fonctionne sur un serveur qui se trouve dans un autre fuseau horaire autre que celui que vous souhaitez utiliser comme référence (par exemple, un serveur localisé en Allemagne utilisera le fuseau horaire « Europe/Berlin »). Si votre serveur est un serveur mutualisé hébergeant plusieurs applications en plus de la vôtre, il apparaît normal de ne pas modifier le fichier « php.ini ».
- l'autre cas de figure qui peut se présenter, et qui est souvent lié au premier : vous n'avez pas les droits nécessaires pour modifier le fichier php.ini par vous-même

Bon, quelle que soit la raison pour laquelle le php.ini n'est pas configuré comme vous le souhaiteriez, vous pouvez vous en sortir en modifiant dynamiquement la « timezone » avec le code PHP suivant :

```
if ( date_default_timezone_get() != 'Europe/Paris' ) {
    date_default_timezone_set('Europe/Paris');
}
```

En PHP, on récupère la date et l'heure courante via une donnée que l'on appelle un « timestamp ». C'est un terme que l'on retrouve en SQL et dans de nombreux langages de programmation. On le traduit en français par le terme « horodatage ».

Pour récupérer le timestamp courant, PHP met à notre disposition la fonction `time()` :

```
echo time(), PHP_EOL; //=> 1501740513
```

Cette donnée n'est pas franchement lisible, en tout cas du point de vue d'un humain. Il s'agit du nombre de secondes qui se sont écoulées entre le 1^{er} janvier 1970 et le moment où le script a été exécuté. Si vous exécutez le bout de code ci-dessus, vous obtiendrez une autre valeur que celle que j'ai obtenue, c'est normal 😊.

La fonction PHP `date()` permet d'obtenir un format de sortie plus lisible, et surtout personnalisable. Si on ne lui donne qu'un paramètre, elle renvoie en sortie la date et l'heure courant, mais on peut lui transmettre une variable de type timestamp en second paramètre, comme on le verra dans un autre exemple. Voici un exemple avec un seul paramètre :

```
echo date('Y-m-d h:i:s'), PHP_EOL; //=> 2017-08-03 08:08:33
```

La fonction PHP `date()` accepte un grand nombre de paramètres, elle est très puissante et très souples d'utilisation. Je vous encourage fortement à lire la documentation de cette fonction sur php.net.

La fonction PHP `strtotime()` est très pratique puisqu'il est possible d'utiliser différents mots clés pour obtenir un timestamp spécifique :

```
echo strtotime("now"), PHP_EOL; //=> 1501740513
echo strtotime("29 july 2017"), PHP_EOL; //=> 1501279200
echo strtotime("1 august 2017"), PHP_EOL; //=> 1501538400
echo strtotime("+1 day"), PHP_EOL; //=> 1501826913
echo strtotime("+1 week"), PHP_EOL; //=> 1502345313
echo strtotime("+1 week 2 days 4 hours 2 seconds"), PHP_EOL; //=> 1502532515
echo strtotime("next Thursday"), PHP_EOL; //=> 1502316000
echo strtotime("last Monday"), PHP_EOL; //=> 1501452000
```

L'exemple « +1day » signifie : plus 1 jour par rapport au timestamp courant, donc par rapport à « now ». A partir de là, vous devinez sans problème la signification des autres exemples.

Les bases de données SQL n'utilisent pas toutes le même format de timestamp. On peut s'appuyer sur les fonctions de PHP pour générer un timestamp adapté à chaque cas :

```
$timestamp = strtotime("1 august 2017 +1 week 2 days 4 hours 2 seconds");
echo date('l dS \o\f F Y h:i:s A', $timestamp) . PHP_EOL;
//=> Thursday 10th of August 2017 04:00:02 AM
echo 'timestamp MySQL => ' . date('Y-m-d h:i:s', $timestamp) . PHP_EOL;
//=> 2017-08-10 04:00:02
echo 'timestamp DB2 => ' . date('Y-m-d h:i:s.000000', $timestamp) . PHP_EOL;
//=> 2017-08-10 04:00:02.000000
```

Je vous encourage à lire la documentation du site php.net relative aux fonctions `date()`, `time()` et `strtotime()`. Cette documentation contient beaucoup d'informations et d'exemples intéressants. Je vous encourage aussi à tester les exemples proposés dans cette documentation, pour vous habituer à la manipulation de ces fonctions.

Quand on travaille avec des dates, on utilise très souvent le format ISO 8601, qui permet de manipuler des dates sous cette forme : "2017-07-27". Quelquefois, les développeurs utilisent des formules telles que : "AAAA-MM-JJ" (pour année, mois, jour), pour indiquer dans quel format de date ils travaillent. Il existe différents formats ISO pour la manipulation de dates, pour un approfondissement autour de cette question, je recommande la lecture de cet article :

<http://www.cl.cam.ac.uk/~mgk25/iso-time.html>

On notera que la fonction `strtotime()` peut aussi nous servir à vérifier la validité d'une date reçue au format ISO (en provenance d'un formulaire, ou d'un fichier CSV, XML, ou autre...). Si la date n'est pas comprise par `strtotime()`, la fonction renverra "false". Si on reçoit une date au format ISO et que l'on souhaite la transmettre à la fonction `strtotime()` pour contrôle, il convient d'être prudent et d'utiliser de préférence le format ISO européen : AAAA-MM-JJ. En effet, le format MM-JJ-AAAA, qui est utilisé dans certains pays anglophones, est interprété bizarrement par `strtotime()`. Je ne rentre pas dans le détail, je vous laisse le soin de faire des tests.

Il arrive que les données relatives à une date nous soient fournies en pièces détachées. Heureusement, la fonction `mktime()` va venir à notre secours :

```
$heure = 10;
$minute = 15;
$seconde = 0;
$mois = 7;
$jour = 27;
$annee = 2017;
$new_date = mktime($heure, $minute, $seconde, $mois, $jour, $annee);
echo $new_date; //=> 1501143300
```

On peut aussi utiliser une combinaison de `mktime()` et `date()` pour obtenir le timestamp correspondant au 1er jour du mois en cours :

```
echo mktime(0, 0, 0, intval(date("m")), 1). PHP_EOL; //=> 1501538400
```

Voici un exemple de fonction PHP permettant de calculer facilement l'écart entre 2 timestamps. Je vous laisse le soin de l'analyser et de la tester :

```
/*
 * Détermine l'écart entre 2 dates
 * @param $timestamp1
 * @param $timestamp2
 * @param string $time_unit
 * @return bool|float
 */
function calculate_time_difference($timestamp1, $timestamp2,
    $time_unit='second') {
    $timestamp1 = intval($timestamp1);
    $timestamp2 = intval($timestamp2);
    if ($timestamp1 && $timestamp2) {
        $time_diff = $timestamp2 - $timestamp1;

        $seconds_in_unit = array(
            'second' => 1,
            'minute' => 60,
            'hour' => 3600,
            'day' => 86400,
            'week' => 604800,
            'month' => 2419200
        );

        if (array_key_exists($time_unit, $seconds_in_unit)) {
            return floor($time_diff/$seconds_in_unit[$time_unit]);
        }
    }
    return false;
}

// Avec quelques tests en prime

$mois_courant = mktime(0, 0, 0, 8, 1);
$mois_precedent = mktime(0, 0, 0, 7, 1);
echo calculate_time_difference($mois_precedent, $mois_courant). PHP_EOL;
//=> 2678400
echo calculate_time_difference($mois_precedent, $mois_courant,
    'hour'). PHP_EOL; //=> 744
echo calculate_time_difference($mois_precedent, $mois_courant,
    'month'). PHP_EOL; //=> 1
```

Je vous ai fourni cette fonction PHP car elle est intéressante d'un point de vue pédagogique.

Mais PHP fournit la fonction date_diff() qui fait à peu près la même chose :

```
$datetime1 = date_create('2017-07-01');
$datetime2 = date_create('2017-08-01');
$interval = date_diff($datetime1, $datetime2);
echo $interval->format('%R%a days') . PHP_EOL; //=> +31 days
```

Les fonctions date_create() et date_diff() sont en fait des alias vers les méthodes de la classe DateTime, et vous pouvez écrire la même chose en syntaxe « full object » :

```
$datetime1 = new DateTime('2017-07-01');
$datetime2 = new DateTime('2017-08-01');
$interval = $datetime1->diff($datetime2);
echo $interval->format('%R%a days') . PHP_EOL; //=> +31 days
```

Pour une vue exhaustive sur les fonctions PHP relatives aux dates :

<http://php.net/manual/fr/ref.datetime.php>

Pour une présentation des possibilités de la classe DateTime :

<http://php.net/manual/fr/datetime.diff.php>

<http://php.net/manual/en/datetime.examples-arithmetic.php>

<http://php.net/manual/fr/class.dateinterval.php>

Si vous avez une date en pièces détachées, et que vous souhaitez vérifier sa validité, vous pouvez utiliser la fonction PHP checkdate() qui renvoie « true » ou « false » selon le résultat. Attention à l'ordre des paramètres, comme la plupart des fonctions PHP relatives aux dates, c'est le mois qui est transmis en premier paramètre, et pas le jour :

checkdate (int \$month , int \$day , int \$year)

<http://php.net/manual/en/function.checkdate.php>

La page suivante est intéressante car elle fournit un raccourci vers différentes synthèses relatives aux formats de date et d'heure :

<http://php.net/manual/fr/datetime.formats.php>

Si vous ne trouvez pas votre bonheur parmi les fonctions PHP existantes, il faut savoir qu'il existe des librairies PHP dédiées à ce sujet comme Chronos et Carbon :

<http://www.dereuromark.de/2016/08/12/chronos-let-there-be-time/>

<https://www.sitepoint.com/suggesting-carbon-with-composer-date-and-time-the-right-way/>

A noter que le livre suivant propose un certain nombre de fonctions de manipulation de dates intéressantes :

PHP 5 in Practice, par Elliot White III et Jonathan Eisenhamer (ed. SAMS, 07/2006)

Voici un exemple de fonction tirée de ce livre :

```
/*
 * L'année est-elle bissextile ?
 * @param $y
 * @return bool
 */
function is_leap_year($y) {
    // If the year is divisible by 4 but not by 100 unless by 400
    return (((($y % 4) == 0) && (((($y % 100) != 0) || ((($y % 400) == 0));
}
```

Allez hop ! Une petite boucle de test :

```
foreach(range(2000, 2020) as $i) {
    if(is_leap_year($i)) {
        echo $i . ' est bissextile'.PHP_EOL;
    } else {
        echo $i . ' n\'est pas bissextile'.PHP_EOL;
    }
}
```

Si vous n'êtes pas familiarisé avec la manipulation de dates, vous vous demandez peut être dans quel cas on peut avoir besoin de déterminer si une année est bissextile. Eh bien, on peut avoir besoin de ce type d'information pour calculer les échéances d'un prêt, ou pour déterminer la date d'échéance d'une facture. Mettre comme date d'échéance le 30 ou le 31 février, je vous garantis que ça fait tâche (c'est même le meilleur moyen de faire peur à vos clients). Et si vous mettez un 28 février sur une année bissextile, ou un 29 février sur une année non bissextile, ça fait tâche également.

4.3.2 Sessions et cookies

Je ne vous l'ai pas dit, mais quand Apache discute avec un navigateur, c'est comme si 2 amnésiques discutaient ensemble. Apache reçoit une requête HTTP en provenance d'un navigateur, il renvoie une réponse sur le même canal de discussion, et l'instant d'après il a tout oublié de la conversation précédente. Et c'est pareil du côté du navigateur qui, une fois qu'il a reçu la réponse du serveur et qu'il l'a traitée, oublie tout de l'échange qui vient de se dérouler. Cette amnésie d'un côté et de l'autre est dûe à la nature du protocole HTTP, qui fonctionne selon un principe appelé « stateless ». On pourrait traduire cela par « faible mémoire ».

Est-ce grave, me direz-vous ? Eh bien oui, assez. Imaginez en effet qu'un utilisateur, après s'être authentifié en entrant sur un site, soit obligé de se réauthentifier à chaque fois qu'il souhaite modifier une information sur le site. Je ne doute pas que cet utilisateur va vous détester assez rapidement. Pour éviter cela, nous allons utiliser la notion de « session ».

Je vous propose de découvrir ce mécanisme de session PHP au travers d'un premier exemple. Saisissez-le dans votre éditeur préféré, puis essayez de l'exécuter en essayant d'utiliser le paramètre « timer » dans l'URL d'exécution, en respectant les valeurs indiquées dans le code :

```
<?php
// démarrage du gestionnaire de session
session_start();

if (isset($_GET['timer'])) {
    switch ($_GET['timer']) {
        case 'start':
            $_SESSION['timer_start'] = time();
            $_SESSION['timer_end'] = null;
            break;
        case 'stop':
            if (isset($_SESSION['timer_end']) || $_SESSION['timer_end'] == null) {
                $_SESSION['timer_end'] = time();
            }
            break;
    }
}

if (isset($_SESSION['timer_start'])) {
    echo 'Timer démarré à : ' . $_SESSION['timer_start'] . '<br>';
    if ($_SESSION['timer_end'] != null) {
        echo 'Timer arrêté à : ' . $_SESSION['timer_end'] . '<br>';
        echo 'Ecart : ' . ($_SESSION['timer_end'] - $_SESSION['timer_start']);
    }
}
}
```

Explications :

Pour que le gestionnaire de session fonctionne, il faut le démarrer via la fonction PHP `session_start()`, que je vous recommande de placer en toute première ligne.

Une fois cette fonction `session_start()` lancée, vous disposez du tableau PHP associatif `$_SESSION`, dans lequel vous pouvez stocker les données que vous voulez. Ce tableau `$_SESSION` fonctionne comme n'importe quel autre tableau associatif, vous pouvez donc utiliser dessus toutes les instructions que vous connaissez déjà (`foreach()`, `isset()`, `array_key_exists()`, etc...).

Si vous exécutez le script une première fois avec une URL de ce type (à adapter en fonction de votre répertoire de travail et du nom de votre script PHP) :

<http://localhost/crud2/session1.php?timer=start>

Vous allez obtenir un affichage qui ressemblera à ceci :

Timer démarré à :1501924356

Vous l'aurez reconnu, il s'agit d'un timestamp tel que nous en avons utilisé dans le chapitre précédent. Votre timestamp sera forcément différent du mien, puisqu'il correspond à la date et à l'heure courante.

Relancez le même script en changeant le paramètre « timer » dans l'URL :

<http://localhost/crud2/session1.php?timer=stop>

Vous obtiendrez quelque chose de ce genre :

Timer démarré à :1501924356

Timer arrêté à :1501924506

Ecart : 150

Bon, il s'est écoulé 150 secondes entre le démarrage de mon « timer » et son arrêt.

J'espère que ce premier exemple vous semble compréhensible. J'aurais pu démarrer par un script plus simple n'utilisant pas la variable globale `$_GET`, mais cela aurait rendu l'exemple nettement moins intéressant. Grâce à la présence de `$_GET`, je peux interagir plus facilement sur les données que je souhaite stocker dans `$_SESSION`.

Il est important de souligner que le contenu de `$_SESSION` est spécifique à chaque utilisateur. Si deux utilisateurs utilisent la même application hébergée sur le même serveur PHP, l'utilisateur A verra dans son tableau `$_SESSION` uniquement les données qui le concernent, il n'aura pas accès au tableau `$_SESSION` de l'utilisateur B. L'utilisateur B n'aura pas non plus connaissance de ce qui se passe chez A. Dans notre exemple précédent, cela signifie que A et B auront des timers différents, sans aucun risque de télescopage.

Mais je vous ai dit précédemment que le navigateur et Apache étaient de grands amnésiques. Alors comment le gestionnaire de sessions de PHP s'y prend-il pour pallier les carences de ces deux grands malades ? Comment fait-il pour dissocier l'utilisateur A de l'utilisateur B et leur affecter les bons jeux de données dans le tableau `$_SESSION` ?

Il faut savoir que la fonction `session_start()` a pour effet d'attribuer à l'utilisateur courant un identifiant unique. Cet identifiant unique va permettre à l'interpréteur PHP de savoir à quel utilisateur correspond le jeu de données contenu dans le tableau `$_SESSION`.

Mais où il est stocké cet identifiant ? Dans un fichier texte dont le nom commence par « sess_ » suivi de l'identifiant attribué à chaque utilisateur. L'emplacement de ces fichiers de session varie d'un stack PHP à l'autre, mais on les trouve généralement dans un sous-répertoire « tmp » se trouvant dans les sous-répertoires du stack PHP. Voici un échantillon ce qui se trouve dans le répertoire « tmp » de mon stack PHP tournant sous XAMPP :

Nom	Modifié le	Type	Taille
sess_ddcre4tngerov5njpiu6ht6q6q	05/08/2017 11:15	Fichier	1 Ko
sess_8o9c90d9a6a3a83v6g1o479d5p	28/07/2017 07:47	Fichier	17 Ko
sess_brdcb7ck24lesaluuk447vpjg7r	26/07/2017 11:40	Fichier	27 Ko
sess_1tfknob18qepr57jo81et6qgk0	25/07/2017 14:10	Fichier	1 Ko
sess_r457u3ttq27q0l7dj5ufjdhf8c	18/07/2017 16:10	Fichier	1 Ko
why.tmp	30/03/2013 13:29	Fichier TMP	1 Ko

Bon, il y a peu de fichiers et cette copie d'écran a été réalisée le 5 août 2017, donc vous devinez facilement quel est le fichier qui me concerne. S'il y avait beaucoup d'utilisateurs connectés à mes applications, il y aurait beaucoup de fichiers. Une solution pour identifier le bon fichier pourrait être d'utiliser la fonction PHP `session_id()`, avec un echo elle m'aurait renvoyé ceci :

id de session : ddcre4tngerov5njpiu6ht6q6q

Ah oui, ça correspond bien à mon fichier « sess_xxx » du 5 août.

Bon, puisqu'on est sûr qu'il s'agit du bon fichier, jetons-y un coup d'œil avec un éditeur :

```
timer_start|i:1501924356;timer_end|i:1501924506;
```

Je vous propose d'utiliser la fonction `var_dump()` sur notre tableau `$_SESSION`, pour comparer :

```
array(2) {
    ["timer_start"]=>
    int(1501924356)
    ["timer_end"]=>
    int(1501924506)
}
```

On voit que le fichier de session contient une représentation condensée du contenu du tableau `$_SESSION`. On dit que les données à l'intérieur du fichier texte sont « *sérialisées* ».

Mais entre deux requêtes http, comment PHP sait-il que je suis bien moi, vu qu'on a dit précédemment que le navigateur et Apache étaient de grands amnésiques. Eh bien, il faut que je vous montre autre chose. Ouvrez les outils de développement de votre navigateur, et positionnez-vous dans la vue « *réseau* » (ou « *network* »). Relancez votre script de test du « *timer* » et observez le contenu de l'onglet « *Cookies* » :

État	Méth...	Fichier	En-têtes	Cookies	Paramètres	Réponse	Délais	Aperçu
200	GET	session1.php		<input type="button" value="Filtrer les cookies"/>				
<div style="border: 1px solid #ccc; padding: 5px;"> <p>▼ Cookies de la requête</p> <ul style="list-style-type: none"> _ga: "GA1.1.502949828.1492791051" _utma: "111872281.502949828.1492791051.1499264503.1499274645.2" _utmz: "111872281.1499264503.1.1.utmc...tmccn=(direct) utmcmd=(none)" PHPSESSID: "ddcre4tngerov5njpiu6ht6q6q" </div>								

Tiens, tiens... Il y a un cookie qui s'appelle `PHPSESSID` et que trouve-t-on dedans ? Le fameux identifiant unique que nous a attribué PHP via la fonction `session_start()`.

C'est donc comme cela que le navigateur et Apache s'y prennent pour conserver la mémoire d'une transaction (d'un échange) entre client et serveur : grâce à un cookie stockant un identifiant de session.

Pour résumer :

- la première fois que la fonction session_start() est appelée, l'interpréteur PHP attribue à l'utilisateur un identifiant de session et crée un fichier « sess_ » avec cet identifiant.
 - o A la fin de ce premier appel du script PHP...
 - Si des données ont été placées dans le tableau \$_SESSION, elles vont être « sérialisées » et stockées dans le fichier de session attribué à l'utilisateur courant
 - Un cookie PHPSESSID contenant l'identifiant de session va être transmis à Apache, qui va l'envoyer au navigateur en même temps que la réponse produite par l'interpréteur PHP (qui sera le plus souvent un flux HTML)
- Lors d'un second appel du script PHP, la fonction session_start() regarde si Apache lui a transmis un cookie PHPSESSID. Si c'est le cas, elle récupère l'identifiant de session qu'il contient, et regarde si le serveur contient un fichier « sess_ » suivi de ce même identifiant. Si c'est le cas, elle récupère le contenu du fichier texte, le « désérialise » et stocke les données dans le tableau \$_SESSION pour que le script PHP puisse les exploiter.

Voilà comment Apache et le navigateur, ces deux grands amnésiques, font pour se souvenir de leurs différents échanges. Enfin j'exagère, en fait ils se rappellent juste de l'identifiant de l'échange, pas des données qui ont été manipulées durant cet échange. Mais c'est suffisant en ce qui nous concerne.

Avant de poursuivre, je vous encourage à faire des tests, en exécutant plusieurs fois le script PHP, et en regardant après chaque échange le contenu du fichier de session.

Vous pouvez aussi faire tourner en parallèle un script plus simple, comme celui-ci :

```
<?php
session_start();
if (isset($_SESSION['visits'])) {
    $_SESSION['visits']++;
} else {
    $_SESSION['visits'] = 1;
}
print 'Vous avez visité cette page '. $_SESSION['visits']. ' fois.';
```

C'est un simple compteur de visites. Si vous observez le contenu du fichier sérialisé, vous constaterez qu'il contient les informations suivantes :

timer_start|i:1501924356;timer_end|i:1501924506;visits|i:6;

Voici un exemple de script intéressant, dans lequel on force une régénération de l'identifiant de session toutes les 30 secondes (pour éviter que la session ne soit piratée et réutilisée à des fins illicites) :

```
session_start();
if (!isset($_SESSION['generated']))
    || $_SESSION['generated'] < (time() - 30)) {
    session_regenerate_id();
    $_SESSION['generated'] = time();
}
```

On peut se servir de l'exemple ci-dessus pour mettre en place un mécanisme qui oblige l'utilisateur à se réauthentifier s'il s'est écoulé trop de temps entre l'action en cours et l'action précédente (cas où l'utilisateur s'est endormi devant son écran, ou plus sérieusement, cas où l'utilisateur s'est absenté en oubliant de fermer sa session, laissant son poste de travail à la portée du premier venu). Dans l'exemple ci-dessous, si l'utilisateur s'est endormi 30 minutes devant son écran, et qu'il se décide enfin à lever le petit doigt, on le redirige vers une page sur laquelle il devra resaisir son identifiant et son mot de passe :

```
session_start();
$stop_destroy = false;
if (!array_key_exists('timeout', $_SESSION)) {
    $stop_destroy = true;
} else {
    if (time() - $_SESSION['timeout'] > 1440) { // 1440 secondes = 30 min
        $stop_destroy = true;
    }
}
if ($stop_destroy) {
    @session_destroy();
    $_SESSION = array();
    session_start();
    echo '<font color="#CC0000">Accès non autorisé, patientez SVP...</font>';
    // redirection automatique au bout de 3 secondes => connexion du manager
    header('Refresh: 3; URL=./index.php');
    exit;
}
// rafraîchissement de l'heure stockée en session
$_SESSION['timeout'] = time();
```

Bon, les utilisateurs apprécieront moyennement ce genre de plaisanterie, mais la sécurité n'est pas une plaisanterie. Vous pouvez éventuellement augmenter le délai, 30 minutes c'est peut-être un peu trop restrictif...

Dans l'exemple qui précède, vous noterez la présence de la fonction `session_destroy()`. Quand on fait appel à cette fonction, il est recommandé de vider également le tableau `$_SESSION`, et si possible de supprimer le cookie de session (ce qui n'est pas réalisé dans notre exemple). Si vous souhaitez en savoir plus, le processus est plutôt bien expliqué dans la page suivante :

<http://php.net/manual/fr/function.session-destroy.php>

Je vous propose de terminer sur un dernier exemple dans lequel on génère un jeton (à partir de données temporelle et d'un petit grain de sel), jeton que l'on stocke en session et que l'on passe en paramètre dans la redirection qui suit. C'est un exemple à caractère pédagogique, qui contient plusieurs techniques intéressantes, que vous pourrez utiliser au moins partiellement dans vos applications :

```

session_start();
if (isset($_SESSION['token'])) {
    if (isset($_GET['token']) && $_GET['token'] != $_SESSION['token']) {
        // si le jeton n'est pas présent dans l'URL, ou s'il est différent
        // alors il s'agit probablement d'une attaque et on stoppe le script
        exit("stop");
    } else {
        echo 'tout va bien';
    }
}

if (!isset($_SESSION['token'])) {
    // régénération de l'ID de session (par précaution)
    session_regenerate_id();

    // Calcul d'un jeton crypté
    $salt = 'XUEESXY'; // Insérez votre "grain de sel" dans cette variable
    // jeton combinant numéro de semaine, "salt" et numéro de jour
    $tokenstr = date('W') . $salt . date('d');
    // cryptage du jeton final
    $token = md5($tokenstr);

    // Stockage du jeton en session
    $_SESSION['token'] = $token;

    // Génération de l'URL de redirection à partir de l'URL courante
    // à laquelle on ajoute le jeton
    $url = 'http://localhost'. $_SERVER['PHP_SELF'] . '?token='.$token;

    // on enregistre et ferme la session en cours avant de procéder à la
    // redirection
    session_write_close();

    // redirection incluant le jeton en paramètre dans l'URL
    header('Location: '.$url);
    // arrêt du script courant
    exit();
}

```

Dans cet exemple, on utilise une technique de cryptage, avec la fonction md5(), pour protéger le jeton. Ce n'est pas une technique inviolable, mais elle peut être utilisée pour crypter un jeton

ponctuel à usage unique, par exemple un jeton que l'on pourrait stocker en champ caché de formulaire.

Attention : on ne recommande pas la fonction md5() pour le cryptage de mot de passe, car elle n'est plus suffisamment sécurisée aujourd'hui, au regard des moyens dont disposent les pirates pour « casser » ce type de cryptage. Pour approfondir le sujet, vous trouverez beaucoup d'infos intéressantes dans les pages suivantes :

<http://php.net/manual/fr/faq.passwords.php#faq.passwords.fasthash>

<http://php.net/manual/fr/book.password.php>

Je termine ce chapitre par quelques informations relatives aux performances. J'ai constaté à plusieurs reprises que certains développeurs utilisaient de manière intensive le tableau `$_SESSION`, en y stockant beaucoup de données (et quand je dis beaucoup, c'est vraiment beaucoup). Or cela peut entraîner à la longue des problèmes de performance. Il faut en effet savoir que, à chaque fois que les phases de sérialisation et de désérialisation des données de `$_SESSION` sont déclenchées, ces phases sont coûteuses en ressources, particulièrement si vous stockez beaucoup de données dans `$_SESSION`. Par ailleurs, les fichiers de session sont de simples fichiers textes, et les accès fichiers en environnement Linux ne sont pas particulièrement véloces (enfin c'est très variable d'un Linux à l'autre). Il est recommandé de ne pas surcharger inutilement le tableau `$_SESSION`, et de déporter le stockage des informations utilisateurs – s'il y en a beaucoup – du côté de la base de données (MySQL ou autre).

Vous noterez que j'ai à peine parlé des cookies, c'est surtout parce que je considère qu'il n'y a pas grand-chose à en dire. Il est possible de créer ses propres cookies avec PHP, en plus du cookie PHPSESSID, mais je vous laisse le soin d'étudier la question si vous êtes confronté à ce besoin, tout ce que vous avez besoin de savoir se trouve sur ces 2 pages :

<http://php.net/manual/fr/function.setcookie.php>

<http://php.net/manual/fr/features.cookies.php>

Et pour un tour d'horizon complet du mécanisme de session de PHP, la meilleure référence se trouve ici :

<http://php.net/manual/fr/book.session.php>

Bon, si vous en avez marre des sessions (ce que je peux comprendre), vous pouvez passer directement au chapitre suivant.

Mais pour les plus curieux d'entre vous, et pour ceux qui voudraient s'orienter vers le métier de Devops, il faut savoir qu'il est possible de modifier la configuration du gestionnaire de session PHP, de manière à stocker les informations de session directement en base de données plutôt que dans des fichiers texte. Cela offre plusieurs avantages :

- Une amélioration notable des performances
- La possibilité de répliquer la base de données sur plusieurs serveurs, pour des questions de sécurité et/ou de performances (load balancing).

Cette reconfiguration se fait au moyen de la fonction PHP `session_set_save_handler()`. On trouvera des exemples de mise en œuvre avec MySQL dans la documentation officielle :

<http://php.net/manual/fr/function.session-set-save-handler.php>

Pour ceux qui seraient amenés à utiliser PHP sur de gros systèmes, tels que la plateforme IBMi, vous trouverez un exemple complet de reconfiguration du gestionnaire de session exploitant la base de données DB2 sur cette page :

<http://www.youngprofessionals.com/wiki/index.php/PHP/DB2Session>

4.4 Les formulaires et les tableaux \$_GET et \$_POST

Au travers des chapitres précédents, nous avons étudié quelques aspects essentiels de la syntaxe du PHP. Nous n'avons pas tout vu, et en particulier nous n'avons pas parlé des classes et des objets. Mais je préfère laisser ce sujet de côté, nous en parlerons plus tard.

Car maintenant que nous connaissons les tableaux PHP, nous pouvons nous intéresser de nouveau au navigateur, et étudier de quelle manière nous pouvons faire dialoguer un navigateur (le client) avec un stack PHP (le serveur). Pour établir ce dialogue, nous allons nous intéresser :

- à la balise « a » du HTML,
- aux formulaires HTML
- aux tableaux PHP \$_GET et \$_POST : ces 2 tableaux sont fournis en standard par l'interpréteur PHP pour récupérer les données transmises par les requêtes HTTP de type GET et POST.

Si vous ne sentez pas à l'aise avec les tableaux HTML (ou avec la balise « a »), je vous conseille de vous reporter au support de cours « HTML5 – Premiers pas ».

4.4.1 Le tableau \$_GET

Pour introduire le sujet, je vous propose de créer un script PHP que vous appellerez « pagelink.php » :

```
<?php
$titre = 'Mon premier échange client-serveur';
$encodage = 'UTF-8';
$message1 = "Page de test pour un échange client-serveur via la balise HTML \"a\"";
$mon_html = <<<BLOC_HTML
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset=\"$encodage\">
        <title>$titre</title>
    </head>
    <body>
        <div class="annonce">
            {$message1}
            <br><br>
            <a href="pagecible.php">Appel de pagecible sans paramètres</a><br>
            <a href="pagecible.php?param1=titi">Appel de pagecible avec 1 paramètre</a><br>
            <a href="pagecible.php?param1=titi&param2=gros-minet">Appel de pagecible avec 2
paramètres</a><br>
        </div>
    </body>
</html>
BLOC_HTML;
echo $mon_html;
```

Créez ensuite, dans le même répertoire, le script PHP « pagetable.php », dont voici le code source :

```
<?php

/* La balise "pre" permet de forcer un affichage en police non proportionnelle
   (très pratique pour afficher le contenu d'un tableau pendant une phase de
   mise au point) */
echo '<pre>';
var_dump($_GET); // return un "dump" de la variable $_GET
echo '</pre>';

// Si $_GET contient au moins 1 paramètre...
if (count($_GET)>0) {
    echo '$_GET contient '.count($_GET). ' paramètres<br>';
} else {
    echo '$_GET ne contient aucun paramètre<br>';
}

// 1ère technique permettant de détecter la présence d'un paramètre dans $_GET
if (isset($_GET['param1'])) {
    echo '$_GET contient le paramètre "param1" qui lui-même contient la valeur : ' .
$_GET['param1']. '<br>';
}

// 2ème technique permettant de détecter la présence d'un paramètre dans $_GET
if (array_key_exists('param2', $_GET)) {
    echo '$_GET contient le paramètre "param2" qui lui-même contient la valeur : ' .
$_GET['param2']. '<br>';
}
```

J'ai inséré des commentaires dans ce code pour vous aider à le comprendre. Si malgré tout, le fonctionnement de certaines instructions PHP vous met en difficulté, je vous encourage à parcourir la documentation du site php.net, dans laquelle vous trouverez le descriptif de chaque fonction, accompagné d'exemples.

Maintenant, à partir de votre navigateur, ou du raccourci proposé par votre IDE, lancez l'exécution de la page « pagetable.php ». L'URL de lancement ressemblera peut être à ceci, si vous avez créé – comme moi – un répertoire « coursphp » dans le répertoire « htdocs » de votre stack PHP, et si vous avez placé votre script PHP dans le sous-répertoire « testlink » :

<http://localhost/coursphp/testlink/pagetable.php>

Voici à quoi ressemble la page dans le navigateur :

Page de test pour un échange client-serveur via la balise HTML "a"

[Appel de pagetable sans paramètres](#)
[Appel de pagetable avec 1 paramètre](#)
[Appel de pagetable avec 2 paramètres](#)

Cliquez sur le premier lien, vous devriez voir apparaître ceci :

```
array(0) {  
}  
  
$_GET ne contient aucun paramètre
```

Cliquez sur le bouton de « retour arrière » de votre navigateur, et cliquez sur le second lien :

```
array(1) {  
    ["param1"]=>  
    string(4) "titi"  
}  
  
$_GET contient 1 paramètres  
$_GET contient le paramètre "param1" qui lui-même contient la valeur : titi
```

Retour arrière, puis clic sur le troisième et dernier lien :

```
array(2) {  
    ["param1"]=>  
    string(4) "titi"  
    ["param2"]=>  
    string(10) "gros-minet"  
}  
  
$_GET contient 2 paramètres  
$_GET contient le paramètre "param1" qui lui-même contient la valeur : titi  
$_GET contient le paramètre "param2" qui lui-même contient la valeur : gros-minet
```

Pour rappel, le dernier lien dans la page a la structure suivante :

[Appel de pageable avec 2 paramètres](pageable.php?param1=titi¶m2=gros-minet)

Vous voyez que dans l'attribut « href » de la balise « a », nous avons 2 paramètres placés derrière le point d'interrogation, et séparés par le symbole « & » :

?param1=titi¶m2=gros-minet

Ce sont bien ces mêmes paramètres que nous retrouvons dans le tableau PHP \$_GET, sous la forme de 2 postes distincts. Et vous voyez que \$_GET est un tableau associatif, au même titre que ceux que nous avons créé dans le chapitre d'introduction dédié à ce sujet.

Cela fait un moment que nous ne nous sommes pas intéressés aux coulisses du navigateur. Jetons un coup d'œil à ce qui se passe dans les outils de développement, et en particulier dans l'onglet « réseau » (ou « network »).

Je vais prendre pour exemple l'onglet « network » de Chrome. Voici ce que j'obtiens après avoir cliqué sur le 3^{ème} et dernier lien de ma page « pagelink.php » :

Name	Status	Type	Initiator	Size	Time	Waterfall	
pagecible.php?param1=titi&p...	200	document	Other	553 B	4 ms		100.00 ms

http://localhost/greta91_2017/testlink/pagecible.php?param1=titi¶m2=gros-minet

J'ai reçu un code retour 200, ce qui signifie que tout s'est bien passé.

Si je clique sur la seule ligne de l'affichage, j'obtiens des informations détaillées sur la requête HTTP qui a été envoyée au serveur, et en particulier sur les paramètres de la requête, paramètres que l'on retrouve présentés en détail tout en bas de la vue « headers ».

The screenshot shows the Network tab in the Chrome DevTools. A single request for "pagecible.php?param1=titi..." is listed with a status of 200. The Headers section shows the full HTTP header with cookies, host, referer, user-agent, and query string parameters. The Response section shows the raw PHP output including var_dump() and \$_GET messages.

Si vous cliquez sur l'onglet « Response », vous allez voir apparaître le code HTML généré par le script PHP. On y retrouve la balise « pre », le résultat produit par la fonction PHP var_dump(), et les messages complémentaires (tronqués dans la copie d'écran) générés par les tests utilisant les fonctions PHP isset() et array_key_exists() :

The screenshot shows the Network tab in the Chrome DevTools with the Response tab selected. The same request for "pagecible.php?param1=titi..." is shown. The Response section displays the raw PHP output, including the var_dump() output and the \$_GET messages.

Vous vous souvenez sans doute que, côté serveur, nous avons la possibilité de consulter le fichier « access.log » d'Apache. Voici un échantillon de ce que vous pouvez y trouver :

```
::1 - - [14/Jul/2017:17:30:12 +0200] "GET
/coursphp/testlink/pagecible.php?param1=titi&param2=gros-minet HTTP/1.1" 200 299
"http://localhost/coursphp/testlink/pagelink.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"

::1 - - [14/Jul/2017:17:34:34 +0200] "GET
/coursphp/testlink/pagecible.php?param1=titi&param2=gros-minet HTTP/1.1" 200 299
"http://localhost/coursphp/testlink/pagelink.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"

::1 - - [14/Jul/2017:17:34:43 +0200] "GET
/coursphp/testlink/pagecible.php?param1=titi&param2=gros-minet HTTP/1.1" 200 299
"http://localhost/coursphp/testlink/pagelink.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
```

Nous venons d'étudier les mécanismes en jeu lors d'un échange client-serveur utilisant la méthode GET du protocole HTTP. Je rappelle en effet que la balise HTML « a » génère d'office – quand on clique dessus – une requête HTTP de type GET.

Nous allons retrouver le même principe avec les requêtes HTTP de type POST, et nous allons l'illustrer avec un exemple de formulaire.

4.4.2 Le tableau \$_POST

Pour partir à la découverte du tableau \$_POST, je vous propose de créer le formulaire suivant dans un fichier « pageform.php » :

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

Destination

Aéroport:

Heure d'embarquement:

Slider avec l'élément input de type range

Choisissez un nombre:

Voici la valeur choisie: 6

Valider

Pour créer ce formulaire, je me suis amusé à utiliser quelques-uns des nouveaux types de balises « input » du HTML5 (les types « search », « number » etc..) , ainsi que la balise « select » (en 3^{ème} position) qui elle, est plus ancienne.

Dans cet exemple, nous avons besoin d'un fichier CSS et de 2 scripts PHP. Je vous propose de les regrouper dans un même sous-répertoire « testform », à l'intérieur de votre projet.

Commençons par créer un fichier « pageform.css », dont voici le code source :

```
body {
    padding-top: 60px;
    padding-bottom: 40px;
    margin-left: 10px;
}
form {
    margin-left: 10px;
}
input, select {
    border-radius: 5px;
    border-color: chocolate;
}
input[type="search"] {
    border-radius: 10px;
}
input:required {
    border-color: purple ;
}
input[data-evt] {
    border-color: silver;
}
```

Dans le même répertoire que le fichier CSS précédent, créons un script PHP « pageform.php », dont voici le code source :

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire</title>
    <link rel="stylesheet" href="pageform.css">
</head>
<body>
<form action="pagecible.php" method="post">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche">
        </p>
        <p>
            <label for="number_field">Exemple de champ "number"</label>
            <input type="number" name="number" id="number_field" required >
        </p>
        <p>
            <label for="select_field">Exemple de champ "select"</label>
            <select name="myselect" id="select_field">
                <option value="A">valeur A</option>
                <option value="B">valeur B</option>
                <option value="C">valeur C</option>
            </select>
        </p>
    </fieldset>
    <fieldset>
        <legend>Destination</legend>
        <p>
```

```

        <label for="airport_code">Aéroport:</label>
        <input type="text" name="airport" id="airport_code"
               list="airports" data-evt="airport" />
    </p>
    <p>
        <label>Heure d'embarquement:</label>
        <input type="datetime-local" name="totime" step=3600
               data-evt="datetime" />
    </p>
</fieldset>
<datalist id="airports">
    <option value="ATL" label="Atlanta">
    <option value="MEM" label="Memphis">
    <option value="LHR" label="London Heathrow">
    <option value="LAX" label="Los Angeles">
    <option value="FRA" label="Frankfurt">
    <option value="PAR" label="Paris">
</datalist>
<fieldset>
    <legend>Slider avec l'élément input de type range</legend>
    <label for="yournumber">Choisissez un nombre: </label>
    <input id="yournumber" name="yournumber" type="range"
           min="1" max="10" step="1" value="6"
           onchange="myOutput.value=this.value;" />
    <br>
    <label>Voici la valeur choisie: </label>
    <output name="myOutput"> 6 </output>
</fieldset>
<input type="submit" name="valid" value="Valider" />
</form>
</body>
</html>

```

Vous aurez sans doute remarqué que dans ce premier script PHP, nous n'utilisons pas les balises PHP. Nous n'en avons pas vraiment besoin dans cette première version, car pour l'instant notre formulaire ne contient pas de logique particulière. Nous aurions d'ailleurs pu créer un fichier « pageform.html », mais pour le coup cela nous aurait privé de la possibilité d'insérer du code PHP, et nous aurions dû tôt ou tard modifier l'extension du fichier en « .php ».

Si vous souhaitez vérifier que l'interpréteur PHP est bien actif malgré tout, vous pouvez insérer les balises PHP et un « echo » envoyant un message quelconque, quelque part entre les balises « body » et « /body ».

Nous intégrerons de la logique PHP dans ce script, dans la version suivante, qui sera présentée dans le prochain chapitre.

Le point important dans ce script, c'est la balise « form » ainsi que ses attributs :

```
<form action="pagecible.php" method="post">
```

L'attribut « action » définit le script PHP qui sera appelé dès que l'utilisateur cliquera sur le bouton « submit » du formulaire.

L'attribut « method » définit le type de requête HTTP qui va être utilisé par le formulaire. Nous avons le choix entre les méthodes GET et POST. Il existe d'autres méthodes au sein du protocole HTTP, comme par exemple PUT et DELETE, mais elles ne sont pas disponibles sur les formulaires HTML.

La méthode GET, c'est strictement la même que la méthode GET utilisée par la balise « a ». Ce qui signifie que, d'un point de vue du PHP, les paramètres sont récupérés dans le tableau `$_GET`.

Voici le code source du script « pageable.php », qui se trouve dans le même répertoire que notre formulaire :

```
<?php

/* La balise "pre" permet de forcer un affichage en police non proportionnelle
   (très pratique pour afficher le contenu d'un tableau pendant une phase de
   mise au point *)
echo '<pre>';
var_dump($_POST); // return un "dump" de la variable $_POST
echo '</pre>';

// Si $_POST contient au moins 1 paramètre...
if (count($_POST)>0) {
    echo '$_POST contient '.count($_POST) . ' paramètres<br>';
} else {
    echo '$_POST ne contient aucun paramètre<br>';
}

// Première technique permettant de détecter la présence d'un paramètre dans $_POST
if (isset($_POST['airport'])) {
    echo '$_POST contient le paramètre "airport" qui lui-même contient la valeur : ' .
$_POST['airport']. '<br>';
}

// Seconde technique permettant de détecter la présence d'un paramètre dans $_POST
if (array_key_exists('totime', $_POST)) {
    echo '$_POST contient le paramètre "totime" qui lui-même contient la valeur : ' .
$_POST['totime']. '<br>';
}

echo '<br>Contenu de $_POST affiché avec une boucle foreach<br>';
echo '<ul>' .PHP_EOL;
foreach($_POST as $postkey=>$postval) {
    echo "<li>{$postkey} => {$postval}</li>" .PHP_EOL ;
}
echo '</ul>' .PHP_EOL;
```

Ce script est très proche de celui que j'avais utilisé dans le chapitre précédent pour présenter le tableau `$_GET`. La différence principale, c'est la boucle « `foreach` » que j'ai ajoutée à la fin du script. Cette boucle « parcourt » le tableau `$_POST` et affiche chaque poste du tableau dans une liste HTML de type « `ul` ».

Après avoir saisi quelques infos « bidon » dans mon formulaire, et cliqué sur le bouton « valider », voici ce que j'obtiens dans la page HTML générée par le script « pagecible.php ».

```
array(7) {  
    ["search"]=>  
        string(4) "test"  
    ["number"]=>  
        string(1) "3"  
    ["myselect"]=>  
        string(1) "B"  
    ["airport"]=>  
        string(3) "MEM"  
    ["totime"]=>  
        string(16) "2017-07-27T05:00"  
    ["yournumber"]=>  
        string(1) "4"  
    ["valid"]=>  
        string(7) "Valider"  
}
```

`$_POST` contient 7 paramètres

`$_POST` contient le paramètre "airport" qui lui-même contient la valeur : MEM

`$_POST` contient le paramètre "totime" qui lui-même contient la valeur : 2017-07-27T05:00

Contenu de `$_POST` affiché avec une boucle foreach

- search => test
- number => 3
- myselect => B
- airport => MEM
- totime => 2017-07-27T05:00
- yournumber => 4
- valid => Valider

Je vous encourage à étudier ce qui se passe du côté des outils de développement du navigateur. L'onglet « network » est riche en informations :

\$_POST contient 7 paramètres
\$_POST contient le paramètre "airport" qui lui-même contient la valeur : MEM
\$_POST contient le paramètre "totime" qui lui-même contient la valeur : 2017-07-21T18:00

Contenu de \$_POST affiché avec une boucle foreach

- search => test de recherche
- number => 3
- myselect => B
- airport => MEM
- totime => 2017-07-21T18:00
- yournumber => 8
- valid => Valider

Vous voyez que nous avons affaire à une requête de type POST et que son code statut est 200. Les autres informations en dessous sont des informations contenues dans la réponse HTTP produite par notre stack PHP. La partie qui m'intéresse le plus, c'est celle qui se situe tout en bas de cette fenêtre d'informations. Faites descendre l'ascenseur jusqu'en bas :

Cache-Control: max-age=0
Connection: keep-alive
Content-Length: 109
Content-Type: application/x-www-form-urlencoded
Cookie: PhpstORM-4488da6a=e4a32db3-ce2b-4103-aa7e-370464f8b706; __utma=111872281.343457310.1498570338.1498570338.1; __utmz=111872281.1498570338.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)
Host: localhost
Origin: http://localhost
Referer: http://localhost/greta91_2017/testform/pageform.php
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36

Form Data

search:	test de recherche
number:	3
myselect:	B
airport:	MEM
totime:	2017-07-21T18:00
yournumber:	8
valid:	Valider

Vous voyez que nous retrouvons les champs de notre formulaire, avec pour chacun d'eux le contenu qui a été envoyé dans la requête HTTP transmise par le navigateur au stack PHP.

```
::1 - - [16/Jul/2017:06:27:17 +0200] "POST /greta91_2017/testform/pagecible.php HTTP/1.1" 200  
774 "http://localhost/greta91_2017/testform/pageform.php" "Mozilla/5.0 (Linux; Android 6.0;  
Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile  
Safari/537.36"
```

Comparons cette requête HTTP de type POST à la requête de type GET que nous avions étudiée au chapitre précédent (que je mets ci-dessous pour rappel) :

```
::1 - - [14/Jul/2017:17:30:12 +0200] "GET  
/coursphp/testlink/pagecible.php?param1=titi&param2=gros-minet HTTP/1.1" 200 299  
"http://localhost/coursphp/testlink/pagelink.php" "Mozilla/5.0 (Linux; Android 6.0; Nexus 5  
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Mobile Safari/537.36"
```

... on voit que dans la requête de type POST, les paramètres correspondant aux champs de formulaire n'apparaissent pas. Ils font pourtant bien partie de la requête, mais dans une requête de type POST, ils sont encapsulés dans la requête et n'apparaissent pas dans l'URL. Or c'est cette URL que nous montre le fichier « access.log ».

Je vous encourage vivement à modifier la balise « form » et en particulier l'attribut « method », en y plaçant la valeur « get » :

```
<form action="pagecible.php" method="get">
```

Rafraîchissez la page de votre formulaire, saisissez quelques données et cliquez de nouveau sur le bouton « valider ». Vous voyez que vous obtenez une requête à rallonge, dans laquelle tous les champs du formulaire sont présents :

http://localhost/greta91_2017/testform/pagecible.php?search=test+de+recherche&number=3&myselect=B&airport=MEM&totime=2017-07-26T05%3A00&yournumber=7&valid=Valider

On retrouve ces mêmes paramètres dans la partie détaillée de la vue « Network » :

▼ **Query String Parameters** [view source](#) [view URL encoded](#)

search: test de recherche
number: 3
myselect: B
airport: MEM
totime: 2017-07-26T05:00
yournumber: 7
valid: Valider

Tiens au fait, la modification de la méthode d'envoi du formulaire a eu une conséquence inattendue. Regardez ce qui se passe dans la page affichée après envoi :

```
array(0) {  
}  
$_POST ne contient aucun paramètre  
  
Contenu de $_POST affiché avec une boucle foreach
```

L'interpréteur PHP n'a trouvé aucune information dans le tableau PHP `$_POST`. C'est normal, toutes les informations se trouvent maintenant dans le tableau `$_GET`, et ce cas de figure n'est pas prévu dans notre script PHP.

Des questions se posent alors :

- dans quel cas faut-il utiliser la méthode GET, dans quel cas faut-il utiliser POST ?
- ne serait-il pas plus simple d'utiliser tout le temps GET ?

En règle générale, on réservera la méthode GET aux formulaires dédiés à la recherche d'informations, sous réserve qu'ils contiennent peu de paramètres : car les requêtes HTTP de type GET ont une longueur maximale qui varie d'un navigateur à l'autre, mais qui tourne généralement autour de 1024 caractères.

Par convention, on utilisera systématiquement la méthode POST dans les cas de figure suivants :

- pour les formulaires de mise à jour, création et suppression,
- pour les formulaires contenant beaucoup de paramètres (pas de limite de longueur avec POST, contrairement à GET)
- pour les formulaires susceptibles de contenir des caractères exotiques (chinois au autres) car les requêtes de type GET ne peuvent pas les transmettre correctement (pour de nombreuses raisons un peu longues à détailler ici), alors que les requêtes POST n'ont aucun souci avec ça.

4.4.3 Un formulaire plus intelligent

Nous allons maintenant étudier comment rendre notre formulaire plus intelligent.

Je vous propose de créer un sous-répertoire « testform2 ».

Créez dans ce nouveau répertoire un nouveau script « index.php ». Dans ce script, nous allons créer un formulaire relativement simple, contenant dans un premier temps seulement 2 champs de saisie (nous le complexifierons plus tard). Ce nouveau formulaire contiendra les champs « nom » et « prenom ».

Voici le code source initial de « index.php » :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire</title>
</head>
<body>
<?php
echo 'Contenu de $_POST<br>';
echo '<pre>';
var_dump($_POST);
echo '</pre>';
?>
<fieldset><legend>Formulaire de test</legend>
<form method="POST">
    <label>Nom
        <input type="text" name="nom" value="">
    </label><br><br>
    <label>Prénom
        <input type="text" name="prenom" value="">
    </label><br><br>
    <input type="submit" name="valid1" value="Valider" >
</form>
</fieldset>
</body>
</html>
```

Plusieurs points à signaler :

- j'ai utilisé une balise « fieldset » associée à une balise « legend », pour obtenir un cadre entourant le formulaire. Ce n'était pas indispensable mais ça rend le formulaire un peu moins moche
- j'ai placé la fonction PHP « var_dump » juste avant le formulaire, pour bénéficier d'un mouchard en temps réel, me permettant de suivre de près l'évolution du contenu du tableau \$_POST (puisque mon formulaire utilise la méthode POST).

Voici ce que l'on obtient lorsque l'on demande l'affichage du formulaire la toute première fois :

Contenu de \$_POST

```
array(0) {  
}
```

Formulaire de test

Nom

Prénom

Voici la même page, et le même formulaire, après avoir validé une première saisie :

Contenu de \$_POST

```
array(3) {  
    ["nom"]=>  
    string(3) "aaa"  
    ["prenom"]=>  
    string(3) "bbb"  
    ["valid1"]=>  
    string(7) "Valider"  
}
```

Formulaire de test

Nom

Prénom

On constate que \$_POST contient 3 postes (nom, prenom et valid1) :

- le poste « nom » contient la valeur que je viens de saisir, soit « aaa »
- le poste « prenom » contient la valeur « bbb »
- le poste « valid1 » contient la valeur que j'ai indiquée dans le champs de saisie, soit « Valider ».

La présence du poste « valid1 » dans le tableau `$_POST` peut sembler ennuyeuse, car son contenu est peu intéressant dans le cas présent. On peut facilement faire disparaître ce poste de tableau, en retirant l'attribut « name » du bouton de saisie, comme ceci :

```
<input type="submit" value="Valider" >
```

Du coup, à la saisie suivante, le tableau `$_POST` ne contient plus que les 2 postes suivants :

Contenu de `$_POST`

```
array(2) {  
    ["nom"]=>  
    string(3) "aaa"  
    ["prenom"]=>  
    string(3) "bbb"  
}
```

Dans quel cas pourrait-on avoir besoin de l'attribut « name » sur un bouton de type « submit » ? Ce serait surtout intéressant dans le cas d'un formulaire contenant 2 boutons de type « submit », par exemple un bouton pour « valider » et un bouton pour « annuler ». Cela donnerait ceci :

```
<input type="submit" name="valid" value="Valider" >  
<input type="submit" name="annul" value="Annuler" >
```

Du coup, le tableau `$_POST` contiendrait soit le poste « valid », soit le poste « annul », en fonction du choix de l'utilisateur. On pourrait dès lors se servir de cette information pour effectuer des actions différentes.

Comme notre formulaire est simple, et qu'il n'autorise que la validation, je vous propose de rester sur un bouton de type « submit » sans attribut « name », donc comme ceci :

```
<input type="submit" value="Valider" >
```

Sur mes deux champs de saisie « nom » et « prenom », j'aimerais définir plusieurs règles de validation, telles que :

- les 2 champs de saisie sont obligatoires
- les 2 champs de saisie ne peuvent contenir de texte de moins de 2 caractères, et ne peuvent contenir de texte de plus de 40 caractères
- si les 2 champs de saisie sont correctement renseignés, je souhaite effectuer une redirection automatique vers un script PHP qui contentera d'afficher les valeurs saisies dans le formulaire, par exemple dans une liste de type « ul ».

Voyons comment intégrer ces différentes contraintes.

Dans le cours « HTML5 Premiers pas », vous aviez vu qu'il était possible d'utiliser un attribut « required » pour rendre les champs de saisie obligatoires. On peut même utiliser des attributs « minlength » et « maxlength » pour définir les longueurs de saisie minimales et maximales autorisées. Cela donnerait ceci :

```
<input type="text" name="nom" value=""  
      required maxlength="20" minlength="2">
```

C'est tentant, et rien ne nous interdit d'utiliser ces attributs, mais l'attribut « required » est apparu avec la norme HTML5, il n'est pas connu des vieux navigateurs, comme les anciennes versions de IE. De plus, comme il s'agit de contrôles effectués côté navigateur, il est très facile pour un utilisateur un peu connaisseur de les supprimer (en passant par les outils de développement et notamment par l'inspecteur d'élément).

Donc, même si on décide d'implémenter ces contrôles côté navigateur, **il est IMPERATIF de les DOUBLER par des contrôles équivalents côté serveur**. Pour pouvoir tester plus facilement la validité de nos contrôles côté serveur, nous n'allons pas utiliser ici les attributs « required », « minlength » et « maxlength ». Libre à vous de les ajouter par la suite, dès que vous serez certain que les contrôles côté serveur sont opérationnels.

Ces mises au point étant faites, revenons à notre formulaire et à son fonctionnement actuel.

Vous avez sans doute constaté un comportement gênant : à chaque fois que l'on soumet notre formulaire, les valeurs saisies disparaissent du formulaire, alors qu'elles sont présentes dans le tableau \$_POST. Ceci est dû au fait que notre formulaire contient des attributs « value » dont la valeur est définie à blanc. Il conviendrait d'insérer dans ces attributs les valeurs contenues dans \$_POST. On peut tester cette solution :

```
<label>Nom  
  <input type="text" name="nom"  
        value="<?php echo $_POST['nom']; ?>">  
</label><br><br>  
<label>Prénom  
  <input type="text" name="prenom"  
        value="<?php echo $_POST['prenom']; ?>">  
</label><br><br>
```

Vous voyez que l'on a effectué un « echo » pour afficher la valeur des postes « nom » et « prenom » du tableau \$_POST.

Réaffichons notre formulaire... nous obtenons ceci :

Contenu de \$_POST

```
array(0) { }
```

Formulaire de test

Nom
Notice:

Prénom
Notice:

Valider

C'est moche ! Nos deux champs de saisie contiennent des messages PHP de type « notice ». J'en ai extrait un par copier-coller pour que vous puissiez voir de quoi il s'agit :

```
<br /><b>Notice</b>: Undefined index: nom in
<b>D:\Tools\xampp\htdocs\GRETA_miniprojet\testform2\index_2.php</b>
on line <b>18</b><br />
```

L'interpréteur PHP nous dit très clairement que nous lui demandons de faire quelque chose d'anormal, à savoir d'afficher un poste de tableau qui n'existe pas. Et pour cause, notre tableau \$_POST est vide lors du tout premier affichage du formulaire.

Comment corriger cela ? On pourrait écrire quelque chose comme ça :

```
<label>Nom
<input type="text" name="nom"
       value=<?php
           if (isset($_POST['nom'])) {
               echo $_POST['nom'];
           } else {
               echo '';
           } ?>">
</label><br><br>
<label>Prénom
<input type="text" name="prenom"
       value=<?php
           if (isset($_POST['prenom'])) {
               echo $_POST['prenom'];
           } else {
               echo '';
           } ?>">
</label><br><br>
```

La fonction PHP `isset()` nous permet de vérifier si le poste existe bien dans le tableau indiqué. C'est un moyen fiable et efficace de régler notre problème. Si vous n'aimez pas la fonction `isset()`, peut être préférez-vous la fonction PHP `array_key_exists()`, qui fait la même chose mais a le mérite d'être plus explicite. Plutôt que d'écrire ceci :

```
if (isset($_POST['nom'])) {
```

... vous pourriez écrire ceci (pour aboutir au même résultat) :

```
if (array_key_exists('nom', $_POST)) {
```

Autre type d'écriture possible, plus compacte, et il est vrai un peu déroutante quand on la découvre pour la première fois :

```
echo isset($_POST['nom']) ? $_POST['nom'] : '';
```

C'est du « tout en un », mais fonctionnellement c'est strictement équivalent au code suivant :

```
if (isset($_POST['nom'])) {
    echo $_POST['nom'];
} else {
    echo '';
}
```

On obtient ainsi dans notre formulaire un code plus compact :

- <label>Nom
`<input type="text" name="nom"`
`value="php echo isset($_POST['nom']) ? $_POST['nom'] : ''; ?>"</code
>
</label>

<label>Prénom
<input type="text" name="prenom"
value="php echo</code
isset($_POST['prenom']) ? $_POST['prenom'] : ''; ?>"
</label>

`

... mais c'est loin d'être satisfaisant. Imaginez que vous deviez écrire ça sur un formulaire composé de 30 champs de saisie... c'est horrible !!! 😞

L'idéal serait de disposer d'un tableau prémâché, contenant soit les valeurs saisies par l'utilisateur, soit des valeurs par défaut, donc des blancs. Il y a plusieurs manières de s'y prendre, une manière assez astucieuse consistant à créer un petit tableau définissant la liste des champs de saisie (en l'occurrence « nom » et « prenom ») et vérifiant la présence de ces champs dans `$_POST`. Si ces champs ne sont pas présents dans `$_POST`, alors on place dans ce tableau

(que j'ai appelé \$datas) des blancs par défaut, pour les postes « noms » et « prenom ». Ce qui nous donne :

```
// Création d'un tableau $datas contenant les données provenant
// de $_POST, ou des valeurs à blanc dans le cas contraire
$dataform = array('nom', 'prenom');
$datas = [];
foreach($dataform as $form) {
    $datas[$form] = isset($_POST[$form]) ? trim($_POST[$form]) : '';
}
```

On peut placer un var_dump sur \$datas pour s'assurer que le tableau est correctement rempli dans tous les cas, y compris lors du tout premier affichage, comme ici :

Contenu de \$_POST

```
array(0) {  
}
```

Contenu de \$datas

```
array(2) {  
    ["nom"]=>  
        string(0) ""  
    ["prenom"]=>  
        string(0) ""  
}
```

Formulaire de test

Nom

Prénom

... et lors d'un second affichage, après avoir saisi des données :

Contenu de \$_POST

```
array(2) {
    ["nom"]=>
        string(3) "aaa"
    ["prenom"]=>
        string(3) "bbb"
}
```

Contenu de \$datas

```
array(2) {
    ["nom"]=>
        string(3) "aaa"
    ["prenom"]=>
        string(3) "bbb"
}
```

Formulaire de test

Nom

Prénom

Juste en dessous de la boucle d'alimentation du tableau \$datas, nous pouvons placer une seconde boucle qui va réaliser un certain nombre d'opérations de nettoyage et de déchappement de caractères problématiques :

```
foreach($datas as $datakey=>$dataval) {
    // suppression des balises HTML saisies dans le formulaire (comme <script>)
    $dataval = strip_tags($dataval);
    // suppression des blancs à droite et à gauche
    $dataval = trim($dataval);
    // échappement des caractères " '...
    $dataval = htmlentities($dataval, ENT_QUOTES, 'UTF-8');
    // $dataval étant nettoyé, mise à jour du tableau d'origine
    $datas[$datakey] = $dataval;
}
```

On améliore ainsi la sécurité et la robustesse du formulaire car :

- L'usage de la fonction strip_tags va nous débarasser d'éventuels balises HTML que l'utilisateur aurait pu saisir dans les champs de formulaire, en particulier la balise « script » qui permet d'insérer du code Javascript potentiellement dangereux.
- L'usage de la fonction htmlentities permet d'échapper certains caractères HTML problématiques avec les formulaires, comme les apostrophes et les guillemets.

Il est temps de supprimer les var_dump sur \$_POST et \$datas, et de les remplacer par un code permettant de contrôler la validité des données saisies, selon les critères que nous avons fixés au début de ce chapitre :

```
// on ne fait que les contrôles que si $_POST contient des données
$erreurs = array();
if (count($_POST)>0) {
    foreach($datas as $datakey=>$dataval) {
        if($dataval == '') {
            $erreurs[$datakey] = 'zone obligatoire';
        } else {
            if (strlen($dataval)>40) {
                $erreurs[$datakey] = 'Longueur maximale autorisée 40c';
            } else {
                if (strlen($dataval)<3) {
                    $erreurs[$datakey] = 'Longueur minimale autorisée 3c';
                }
            }
        }
    }
}

if(count($erreurs)>0) {
    echo '<fieldset><legend>Liste des erreurs</legend>';
    echo '<ul>';
    foreach($erreurs as $errkey=>$errval) {
        echo '<li>' . $errkey . ' : ' . $errval . '</li>';
    }
    echo '</ul>';
    echo '</fieldset>';
}
```

Voici un échantillon de tests :

Liste des erreurs

- nom : Longueur maximale autorisée 40c
- prenom : Longueur minimale autorisée 3c

Formulaire de test

Nom

Prénom

Liste des erreurs

- nom : Longueur minimale autorisée 3c
- prenom : zone obligatoire

Formulaire de test

Nom

Prénom

A ce stade, nous pouvons mettre en place la redirection vers un autre script, que j'ai appelé « cible.php ». Voici ce que l'on pourrait être tenté d'y écrire :

```
<?php
echo '<br>Contenu de $_POST affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_POST as $postkey=>$postval) {
    echo "<li>{$postkey} => {$postval}" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Mais cela ne va pas fonctionner, car dans le script « cible.php », la variable `$_POST` ne contiendra aucune donnée intéressante pour nous, en réalité il sera vide. Il nous faut un autre moyen de transmettre les données contenues dans notre tableau `$datas`. Et ce moyen consiste à passer par le tableau `$_SESSION`. Et pour cela, il faut que la fonction « `session_start` » soit

placé au début des 2 scripts (index.php et cible.php). De plus à l'intérieur de « index.php », nous allons placer le code suivant après le code de contrôle des erreurs :

```
// redirection effectuée si pas d'erreurs, avec stockage préalable des données
// dans $_SESSION
if(count($erreurs)==0 && count($_POST)>0) {
    $_SESSION['redirection'] = $datas;
    header('location: cible.php');
    exit();
}
```

Et dans le script cible.php, nous allons placer le code suivant :

```
<?php
session_start();

echo '<ul>';
foreach ($_SESSION['redirection'] as $key=>$value) {
    echo '<li>' . $key . ' : ' . $value . '</li>';
}
echo '</ul>';
```

J'ai aussi un peu remanié le script index.php, de manière à mettre en place la redirection avant d'envoyer quoi que ce soit dans le buffer HTML. En effet, il est dangereux de vouloir effectuer une redirection si des données ont déjà été envoyées dans le buffer HTML, cela peut provoquer des dysfonctionnements.

Voici la nouvelle version du script index.php :

```
<?php
session_start();

// Création d'un tableau $datas contenant les données provenant
// de $_POST, ou des valeurs à blanc dans le cas contraire
$dataform = array('nom', 'prenom');
$datas = [];
foreach($dataform as $form) {
    $datas[$form] = isset($_POST[$form]) ? trim($_POST[$form]) : '';
}

foreach($datas as $datakey=>$dataval) {
    // suppression des balises HTML saisies dans le formulaire (comme <script>)
    $dataval = strip_tags($dataval);
    // suppression des blancs à droite et à gauche
    $dataval = trim($dataval);
    // échappement des caractères " '...
    $dataval = htmlentities($dataval, ENT_QUOTES, 'UTF-8');
    // $dataval étant nettoyé, mise à jour du tableau d'origine
    $datas[$datakey] = $dataval;
}

// on ne fait que les contrôles que si $_POST contient des données
$erreurs = array();
```

```

if (count($_POST)>0) {
    foreach($datas as $datakey=>$dataval) {
        if($dataval == '') {
            $erreurs[$datakey] = 'zone obligatoire';
        } else {
            if (strlen($dataval)>40) {
                $erreurs[$datakey] = 'Longueur maximale autorisée 40c';
            } else {
                if (strlen($dataval)<3) {
                    $erreurs[$datakey] = 'Longueur minimale autorisée 3c';
                }
            }
        }
    }
}

// redirection effectuée si pas d'erreurs, avec stockage préalable des données dans
// SESSION
if(count($erreurs)==0 && count($_POST)>0) {
    $_SESSION['redirection'] = $datas;
    header('location: cible.php');
    exit();
}
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire</title>
</head>
<body>
<?php
if(count($erreurs)>0) {
    echo '<fieldset><legend>Liste des erreurs</legend>';
    echo '<ul>';
    foreach($erreurs as $errkey=>$errval) {
        echo '<li>' . $errkey . ' : ' . $errval . '</li>';
    }
    echo '</ul>';
    echo '</fieldset>';
}
?>
<fieldset><legend>Formulaire de test</legend>
<form method="POST">
    <label>Nom
        <input type="text" name="nom"
               value="php echo $datas['nom']; ?&gt;"&gt;
    &lt;/label&gt;&lt;br&gt;&lt;br&gt;
    &lt;label&gt;Prénom
        &lt;input type="text" name="prenom"
               value="<?php echo $datas['prenom']; ?&gt;"&gt;
    &lt;/label&gt;&lt;br&gt;&lt;br&gt;
    &lt;input type="submit" value="Valider" &gt;
&lt;/form&gt;
&lt;/fieldset&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre

```

TODO : à finaliser (avec un champ de type email)

- J'ai aussi ajouté une option au champ « select » qui est la suivante, je l'ai placée en première position :


```
<option value="">Choisissez une valeur</option>
```
- Dans le formulaire, vous aurez peut être remarqué que j'ai modifié la balise « form » de la façon suivante :


```
<form method="POST">
```

Explication :

- je souhaite utiliser la méthode POST, considérant que mon formulaire va être utilisé pour effectuer de la mise à jour côté serveur (même si ce n'est pas vrai pour l'instant)
- le fait d'avoir supprimé l'attribut « action » va avoir pour effet que le navigateur va « boucler » sur la même page, donc sur le même script « pageform.php ». Nous devrons donc trouver un moyen pour lancer l'exécution du script « pagecible.php » (c'est bien sûr PHP qui va nous aider dans cette tâche, nous verrons comment dans un moment).

Maintenant, je vais instaurer quelques nouvelles règles de fonctionnement pour notre formulaire :

1. tous les champs de saisie sont obligatoires, et je souhaite que le contrôle soit effectué par PHP (d'où la suppression de l'attribut « required » sur l'un des champs de saisie)
2. c'est seulement quand tous les champs de saisie seront bien remplis, que je déclencherais l'appel du script « pagecible.php ». Tant que certains champs de saisie resteront « vides », je bouclerai sur le formulaire en cours.
3. Lors du réaffichage du formulaire, les champs correctement renseignés par l'utilisateur devront conserver la valeur saisie pour ne pas obliger l'utilisateur à les saisir de nouveau.

Nous allons voir que ces règles en apparence simples vont avoir un certain impact sur le code de notre script « pageform.php ».

Pour comprendre comment agir sur notre script de gestion de formulaire, je vous propose de modifier le script « pageform.php », en ajoutant le code suivant, juste en dessous de la balise « body », et juste avant la balise « form » :

```
<?php
echo 'Dump de $_POST :<br>';
echo '<pre>';
var_dump($_POST);
```

```
echo '</pre><br>';
?>
```

Vous connaissez déjà la fonction var_dump(), je ne la présente pas ici. Je rappelle simplement que la balise « pre », influe sur la manière dont le navigateur affiche les infos renvoyées par var_dump(). Vous pourrez vous amuser à supprimer la balise « pre » pour voir la différence avant/après. La balise « /pre » placée après le var_dump() a pour effet de remettre le navigateur dans un mode d'affichage normal.

Mais revenons à nos moutons... Maintenant que nous avons ajouté la fonction var_dump(), je vous propose de regarder ce que ça donne dans le navigateur :

Dump de \$_POST :

```
array(0) { }
```

— Formulaire en vrac —

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

La fonction var_dump() nous indique clairement que le tableau \$_POST est vide au premier affichage de la page.

OK, saisissez maintenant quelques valeurs dans le formulaire, comme par exemple ceci :

Dump de \$_POST :

```
array(0) {}
```

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

... puis cliquez sur « valider » :

Dump de \$_POST :

```
array(4) {  
    ["search"]=>  
    string(16) "ceci est un test"  
    ["number"]=>  
    string(1) "4"  
    ["myselect"]=>  
    string(0) ""  
    ["valid"]=>  
    string(7) "Valider"  
}
```

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

Intéressant non ? Le tableau \$_POST est maintenant bien rempli. Vous notez que j'ai laissé le champ « select » avec sa première option (dont la valeur est à blanc). Il apparaît bien comme tel dans le tableau.

Par contre, dans le formulaire, tous les champs sont réinitialisés, nous avons perdu notre saisie. Ou plus exactement, nous avons déporté cette saisie dans le tableau `$_POST`, mais nous n'avons rien fait pour que le formulaire soit en mesure de réafficher les valeurs saisies. Nous devons lui ajouter un peu d'intelligence.

Une première manière de faire consiste à ajouter un peu de code PHP à l'intérieur de chaque champ de saisie, au niveau de l'attribut « value », et d'y placer le contenu des postes « search », « number » et « myselect ». Dans l'exemple ci-dessous, j'ai « surligné » en gris plus foncé les parties modifiées :

```
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche"
                   value="<?php echo $_POST['search']; ?>">
        />
    </p>
    <p>
        <label for="number_field">Exemple de champ "number"</label>
        <input type="number" name="number" id="number_field"
               value="<?php echo $_POST['number']; ?>">
    />
    </p>
    <p>
        <label for="select_field">Exemple de champ "select"</label>
        <select name="myselect" id="select_field"
               value="<?php echo $_POST['myselect']; ?>">
            <option value="">Choisissez une valeur</option>
            <option value="A">valeur A</option>
            <option value="B">valeur B</option>
            <option value="C">valeur C</option>
        </select>
    </p>
    </fieldset>
    <input type="submit" name="valid" value="Valider" />
</form>
```

Regardons page suivante ce que cela donne côté navigateur.

Dump de \$_POST :

```
array(0) {
}
```

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

Oups, c'est bizarre ce qui se produit dans le champ de recherche. On ne le voit pas sur le champ « number », car comme c'est champ de type « number », le navigateur masque certaines infos, mais le code source HTML est sans appel... ça va mal sur les 3 champs de saisie !

```
<body>
Dump de $_POST :<br><pre>array(0) {
}</pre><br><form method="POST">
<fieldset>
    <legend>Formulaire en vrac</legend>
    <p>
        <label for="search_field">Exemple de champ "search"</label>
        <input type="search" name="search" id="search_field"
            placeholder="saisissez un critère de recherche"
            value="<br />"/>
    <b>Notice</b>: Undefined index: search in <b>D:\Tools\xampp\htdocs\GRETA91_2017\testform2\pageform.php</b> on line <b>21</b><br />
    </p>
    <p>
        <label for="number_field">Exemple de champ "number"</label>
        <input type="number" name="number" id="number_field"
            value="<br />"/>
    <b>Notice</b>: Undefined index: number in <b>D:\Tools\xampp\htdocs\GRETA91_2017\testform2\pageform.php</b> on line <b>27</b><br />
    </p>
    <p>
        <label for="select_field">Exemple de champ "select"</label>
        <select name="myselect" id="select_field"
            value="<br />">
            <option value="">Choisissez une valeur</option>
            <option value="A">valeur A</option>
            <option value="B">valeur B</option>
            <option value="C">valeur C</option>
        </select>
    </p>
</fieldset>
<input type="submit" name="valid" value="Valider" />
</form>
</body>
```

On voit que sur chaque champ de saisie, l'attribut « value » contient un message de type « notice » envoyé par PHP, nous indiquant que nous lui demandons de faire n'importe quoi.

Bon, quand on y réfléchit, c'est assez logique. Lors du tout premier affichage de la page, on demande au PHP d'insérer dans le code HTML des informations qui n'existent pas encore dans le tableau `$_POST`. Ces informations ne seront présentes dans `$_POST` qu'à partir du moment où nous aurons cliqué sur le bouton « valider ». Il nous faut donc insérer du code PHP un peu plus intelligent.

Donc plutôt que d'écrire ceci :

```
<input type="search" name="search" id="search_field"
       placeholder="saisissez un critère de recherche"
       value="php echo $_POST['search']; ?&gt;" /&gt;</pre

```

... je vous propose d'écrire cela :

```
<input type="search" name="search" id="search_field"
       placeholder="saisissez un critère de recherche"
       value="php echo isset($_POST['search']) ? $_POST['search'] : ''; ?&gt;" /&gt;</pre

```

Explication : cette syntaxe bizarre que j'ai utilisée dans l'exemple précédent, et que je vous remets ci-dessous...

```
isset($_POST['search']) ? $_POST['search'] : '';
```

... revient en définitive à écrire ceci (en syntaxe algorithmique) :

```
Si le poste « search » existe dans le tableau $_POST
Alors je renvoie en sortie le contenu de $_POST['search']
Sinon je renvoie en sortie un blanc
FinSi
```

C'est en fait un test en condensé, j'aurais pu écrire un vrai test « if »... comme ceci :

```
<input type="search" name="search" id="search_field"
       placeholder="saisissez un critère de recherche"
       value="php if (isset($_POST['search'])) echo $_POST['search']; else
echo ''; ?&gt;" /&gt;</pre

```

... cela aurait fonctionné également, ça faisait juste un peu plus de code à saisir. En termes de lisibilité, je préfère la version condensée, mais dans un cas comme dans l'autre, il faut bien reconnaître que ça fait un code assez... moche (pour rester poli). Et en plus il va falloir le dupliquer sur chacun des champs... berk !!! :Q

Mais au fait, est-ce que ce code fonctionne dans tous les cas ? Je ne m'inquiète pas trop pour le champ de type « number », mais est-ce que cela va réellement fonctionner avec le champ de type « select » ? Le mieux c'est d'essayer non ?

Voici le code PHP inséré dans le code HTML du champ « myselect » :

```
<select name="myselect" id="select_field"
        value=<?php echo isset($_POST['myselect']) ? $_POST['myselect'] : '';
?>>
```

Et voici le même code en fonctionnement :

The screenshot shows the browser's developer tools with the DOM tree on the right and the element inspector on the left. The element being inspected is a dropdown menu with the ID 'select_field'. The DOM tree shows the following structure:

```
Dump de $_POST :"
<br>
▶ <pre>...</pre>
<br>
▼ <form method="POST">
  ▼ <fieldset>
    <legend>Formulaire en vrac</legend>
    ▶ <p>...</p>
    ▶ <p>...</p>
    ▶ <p>...</p>
    ▶ <p>...</p>
    <label for="select_field">Exemple de champ "select"</label>
    ... ▼ <select name="myselect" id="select_field" value="B" == $0
      <option value="">Choisissez une valeur</option>
      <option value="A">valeur A</option>
      <option value="B">valeur B</option>
      <option value="C">valeur C</option>
    </select>
  </fieldset>
  <input type="submit" name="valid" value="Valider">
</form>
</body>
</html>
```

The element inspector on the left shows the following state for the dropdown:

- Value: B (highlighted in blue)
- Label: Exemple de champ "select"
- Options:
 - Choisissez une valeur
 - valeur A
 - valeur B (highlighted in blue)
 - valeur C

C'est bizarre non ? L'inspecteur d'élément nous indique que l'attribut « value » est bien alimenté avec la valeur sélectionnée (en l'occurrence « B »), pourtant c'est toujours la première option qui s'affiche (et qui correspond au code option à blanc).

Bon, si vous avez zappé la partie de mon cours « HTML5 – premiers pas » relative au champ « select » (ou si vous l'avez lu mais un peu oublié, auquel cas je vous pardonne 😊), je rappelle ce que j'indiquais dans le cours HTML5 :

Exemple de liste déroulante :

```
<select name="test">
  <option value="0">Développeur</option>
  <option value="1" selected>Utilisateur</option>
</select>
```

Exemple de liste déroulante : ▾

L'attribut « *selected* » est facultatif. Quand il est défini, il détermine quelle est la valeur sélectionnée par défaut.

Bref, en ce qui concerne notre code... c'était bien essayé 😊, mais ça ne pouvait pas fonctionner. Une solution fonctionnelle, en revanche, ce serait celle-ci :

```
<select name="myselect" id="select_field">
    <option value="">Choisissez une valeur</option>
    <option value="A" <?php echo isset($_POST['myselect']) &&
        $_POST['myselect'] == 'A'? 'selected': ''; ?>>valeur A</option>
    <option value="B" <?php echo isset($_POST['myselect']) &&
        $_POST['myselect'] == 'B'? 'selected': ''; ?>>valeur B</option>
    <option value="C" <?php echo isset($_POST['myselect']) &&
        $_POST['myselect'] == 'C'? 'selected': ''; ?>>valeur C</option>
</select>
```

La vache !!! 😱, ça commence à piquer les yeux tout ce code PHP disséminé dans le code HTML. C'est « pas glop », dirait le Pifou 😊. J'en connais d'autres qui diraient que ça sent le mois. Je vous invite quand même à vérifier que le code fonctionne (moi j'ai déjà fait le test, c'est bon en ce qui me concerne).

En fait, il existe plein de manières différentes de générer des formulaires à partir de PHP. Il existe plusieurs styles, plusieurs écoles pourrait-on dire. Je ne vais certainement pas contenter tout le monde (je ne vais même pas essayer), mais nous allons étudier 2 approches différentes qui peuvent toutes deux convenir, au moins pour les cas relativement simples comme l'est notre formulaire de test.

La première approche va consister à créer deux fonctions PHP, qui permettront d'alléger notre code HTML. Nous allons l'étudier dans la suite de ce chapitre.

La seconde approche va consister à générer la totalité d'un champ de saisie HTML à partir de code PHP. Cela nous demandera un peu plus de travail, nous l'étudierons au chapitre 4.4.5 (« générateur de formulaire »).

Voyons cette première approche, qui consiste à simplifier l'écriture du code PHP imbriqué dans le code HTML, sans pour autant tout « péter ».

Je vais prendre quelque raccourcis, en vous proposant deux fonctions toutes faites, que je vous laisse le soin d'étudier (le code n'est pas très compliqué) :

- La fonction `get_form_input_value()`, à utiliser sur les 2 premiers champs du formulaire
- La fonction `get_form_select_selected()`, à utiliser sur le champ de type « select »

```
/**
 * Renvoie en sortie la valeur d'un champ de saisie
 * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
 *   'post' (valeur par défaut)
 *   'get'
 * @param $field
 * @param string $method
 * @return string
 */
function get_form_input_value($field, $method='post') {
    // suppression des blancs parasites et forçage des valeurs en minuscule
    $field = strtolower(trim($field));
    $method = strtolower(trim($method));

    if ($method == 'get') {
        $value = isset($_GET[$field]) ? trim($_GET[$field]): '';
    } else {
        $value = isset($_POST[$field]) ? trim($_POST[$field]): '';
    }
    return $value;
}

/**
 * Renvoie en sortie la valeur "selected" pour les champs de type SELECT
 * si la valeur du champ considéré est égale au second paramètre
 * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
 *   'post' (valeur par défaut)
 *   'get'
 * @param $field
 * @param $option_key
 * @param string $method
 * @return string
 */
function get_form_select_selected($field, $option_key, $method='post') {
    // suppression des blancs parasites et forçage des valeurs en minuscule
    $field = strtolower(trim($field));
    $method = strtolower(trim($method));
    if ($method == 'get') {
        $value = isset($_GET[$field]) ? trim($_GET[$field]): '';
    } else {
        $value = isset($_POST[$field]) ? trim($_POST[$field]): '';
    }
    if ($value == $option_key) {
        return 'selected';
    }
    return '';
}
```

Je vous propose de placer la déclaration de ces 2 fonctions dans le script « pageform.html », juste après l'appel de la fonction var_dump().

Voyons maintenant comment placer l'appel de ces fonctions dans le formulaire :

```
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche"
                   value=<?php echo get_form_input_value('search'); ?>">
        />
    </p>
    <p>
        <label for="number_field">Exemple de champ "number"</label>
        <input type="number" name="number" id="number_field"
               value=<?php echo get_form_input_value('number'); ?>">
    />
    </p>
    <p>
        <label for="select_field">Exemple de champ "select"</label>
        <select name="myselect" id="select_field">
            <option value="">Choisissez une valeur</option>
            <option value="A" ><?php echo get_form_select_selected('myselect', 'A'); ?>>valeur A</option>
            <option value="B" ><?php echo get_form_select_selected('myselect', 'B'); ?>>valeur B</option>
            <option value="C" ><?php echo get_form_select_selected('myselect', 'C'); ?>>valeur C</option>
        </select>
    </p>
    </fieldset>
    <input type="submit" name="valid" value="Valider" />
</form>
```

Bon, ce n'est pas le Pérou, mais c'est déjà mieux. Mais j'entends déjà un de mes élèves au fond de la classe : « eh m'sieur, on appelle 3 fois la fonction get_form_select_selected(), ce n'est pas un peu redondant ? »

Je suis d'accord, on peut effectivement faire mieux.

On peut par exemple écrire quelque chose de ce genre :

```
<select name="myselect" id="select_field">
    <option value="">Choisissez une valeur</option>
    <?php
    foreach(array('A', 'B', 'C') as $valselect) {
        echo "<option value=\"{$valselect}\">\n"
        . get_form_select_selected('myselect', $valselect)
        . ">valeur {$valselect}</option>" . PHP_EOL;
    };
    ?>
</select>
```

Dans cet exemple, j'ai généré dynamiquement un tableau contenant les valeurs A, B et C, et je m'en suis servi pour exécuter une boucle « foreach » générant une ligne d'option HTML pour chaque poste du tableau.

Attention, le code ci-dessus est une proposition, qui a le mérite de fonctionner. Mais on peut trouver bien d'autres manières de faire.

Ceci étant, nous n'avons pas couvert toutes les règles de fonctionnement que nous nous étions fixées. Je les rappelle ci-dessous :

1. tous les champs de saisie sont obligatoires, et je souhaite que le contrôle soit effectué par PHP (d'où la suppression de l'attribut « required » sur l'un des champs de saisie)
2. c'est seulement quand tous les champs de saisie seront bien remplis, que je déclencherai l'appel du script « pagecible.php ». Tant que certains champs de saisie resteront « vides », je bouclerai sur le formulaire en cours.
3. Lors du réaffichage du formulaire, les champs correctement renseignés par l'utilisateur devront conserver la valeur saisie pour ne pas obliger l'utilisateur à les saisir de nouveau.

Nous couvrons bien fonctionnellement la 3^{ème} exigence, mais il nous manque les 2 premières.

Pour la gestion des champs de saisie obligatoires, il y a là encore plusieurs manières de faire. On peut même dire qu'il y a deux grandes écoles :

- Soit on affiche toutes les erreurs dans un bloc de commentaires placé au dessus du formulaire
- Soit on affiche les erreurs au niveau de chaque champ de saisie concerné : dans ce cas, on place généralement le message d'erreur à droite de la balise « span », ou entre la balise « span » et la balise « input » concernée, ou quelquefois sous la balise « input ». Le choix de l'emplacement du message d'erreur dépend souvent de contraintes de place, mais aussi du type d'ergonomie recherchée, ou encore des habitudes des utilisateurs, qu'on essaie de ne pas trop perturber, quand c'est possible.

Si on décide de regrouper tous les messages d'erreur dans un bloc de commentaires situé au dessus du formulaire, alors une boucle de type « foreach » peut suffire. On pourrait faire cette boucle directement à partir du tableau \$_POST, et afficher les erreurs au fur et à mesure, mais ce n'est pas une bonne idée, et cela pour plusieurs raisons :

- On peut souhaiter ignorer certaines données contenues dans \$_POST, c'est le cas notamment du poste « valid » qui correspond au bouton de validation du formulaire, et qui ne nous intéresse pas. Il serait plus pratique d'effectuer une première boucle qui filtre \$_POST et ne retient que les données significatives contenues ce tableau.
- On peut souhaiter connaître à l'avance le nombre d'erreurs, pour optimiser l'affichage. Et pour cela, il est plus facile de procéder d'abord à l'analyse de \$_POST, de manière à pouvoir stocker les éventuelles erreurs dans un tableau secondaire. On utilisera ce tableau secondaire pour déterminer le nombre d'erreurs, et bien sûr les afficher.

Concrètement, j'aimerais obtenir un affichage de ce type :

Liste des erreurs

- Champ search : zone obligatoire
- Champ number : zone obligatoire
- Champ myselect : zone obligatoire

Formulaire en vrac

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select"

Voici le code PHP, à insérer en remplacement de la fonction var_dump(), cette dernière ne nous étant plus daucune utilité :

```
// Initialisation du tableau des erreurs
$erreurs = [];
// Boucle pour analyser si des erreurs sont présentes
foreach($_POST as $keypost=>$valpost) {
    if ($keypost != 'valid') {
        $valpost = trim($valpost);
        if ($valpost == '') {
            $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
        }
    }
}
// Si des erreurs sont présentes, alors on en affiche la liste
if (count($erreurs)>0) {
    echo "<fieldset style='background-color: bisque; border-radius: 5px;'>".PHP_EOL;
    echo "<legend>Liste des erreurs</legend>".PHP_EOL;
    echo "<ul>".PHP_EOL;
    foreach($erreurs as $valerr) {
        echo "<li>$valerr</li>".PHP_EOL;
    }
    echo "</ul>".PHP_EOL;
    echo "</fieldset>".PHP_EOL;
}
```

Je vous recommande de tester ce code, de le décortiquer, de regarder aussi avec la fonction d'examen du navigateur le code HTML généré. Réfléchissez aussi à la manière dont vous pourriez améliorer ce code PHP. Vous pouvez envisager de le placer dans une fonction PHP « maison ». Si vous écrivez cette fonction de façon suffisamment générique, vous pourrez la réutiliser sur plusieurs scripts PHP.

Je rappelle qu'il restait une dernière exigence, celle consistant à « débrancher » vers le script « pagecible.php », si le formulaire ne contient plus d'erreurs. Pour cela, il nous faut une fonction de redirection, permettant d'interrompre le script PHP en cours, et de réorienter le flux d'exécution du PHP vers un autre script.

Pour ce faire, PHP met à notre disposition la fonction header().

Elle est pratique cette fonction header(), c'est une sorte de couteau suisse car elle permet de faire beaucoup de choses, mais ce qui nous intéresse ici, c'est la redirection.

Donc, pour effectuer une redirection vers notre script « pagecible.php », il nous suffit d'écrire ceci :

```
header('location: pagecible.php');
exit; // on stoppe l'exécution du script courant
```

Bon, en pratique, c'est légèrement plus compliqué, car nous devons déterminer s'il n'y a plus d'erreurs avant de procéder à la redirection, donc nous pourrions envisager un code de ce type (attention, ce n'est qu'une version provisoire) :

```
if (count($erreurs)==0) {
    // redirection vers "pagecible.php"
    header('location: pagecible.php');
    exit; // on stoppe l'exécution du script courant
}
```

On pourrait placer ce code en complément du code précédent, juste après la boucle d'analyse de \$_POST qui effectue le remplissage du tableau \$erreurs.

Attention, la documentation officielle de la fonction header() indique ceci :

*header() permet de spécifier l'en-tête HTTP string lors de l'envoi des fichiers HTML. /.../
N'oubliez jamais que header() doit être appelée avant que le moindre contenu ne soit envoyé, soit par des lignes HTML habituelles dans le fichier, soit par des affichages PHP. Une erreur très classique est de lire un fichier avec include ou require, et de laisser des espaces ou des lignes vides, qui produiront un affichage avant que la fonction header() ne soit appelée. Le même problème existe avec les fichiers PHP/HTML standards.*

Source : <http://php.net/manual/fr/function.header.php>

Ce qui signifie que, en théorie, il conviendrait de placer notre redirection bien avant d'avoir généré le moindre code dans le buffer HTML (dont je rappelle qu'il commence à se remplir dès qu'on envoie du HTML brut, comme notre entête de page). Mais en pratique, je ne rencontre pas ce problème de redirection sur notre exemple, qui pourtant ne respecte pas la règle édictée ci-dessus (en rouge). J'en viens même à me demander si ce problème est réellement d'actualité, et si la doc de php.net est véritablement à jour sur cette fonction header(). Je conserve néanmoins le souvenir d'avoir rencontré ce problème de redirection inopérante sur certains

environnements PHP sous Linux. On va laisser ce sujet de côté, retenez simplement que, si vous recontrez le problème, il conviendra de déplacer le code à l'origine de la redirection avant tout envoi de donnée vers le buffer HTML, que ce soit du code HTML brut, ou du code généré par les fonctions PHP echo() ou print().

Je rappelle ci-dessous le code source du script « pagecible.php » :

```
<?php
echo '<br>Contenu de $_POST affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_POST as $postkey=>$postval) {
    echo "<li>{$postkey} => {$postval}</li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Je vous propose d'exécuter notre formulaire, de provoquer volontairement une erreur (en oubliant de remplir un champ), pour vérifier que la boucle sur le formulaire fonctionne, puis de remplir tous les champs et de recliquer sur le bouton « valider ». Vous allez voir apparaître le message suivant :

Contenu de \$_POST affiché avec une boucle foreach

Mince ! \$_POST est vide ? Où sont passées mes données? WTF !!!

Ben oui, vous ne pensiez quand même pas vous en tirer si facilement ! 😊
Ce qu'il faut comprendre ici, c'est que le contenu de \$_POST, de même que celui de \$_GET, n'est accessible que dans la requête HTTP initiale. En cas de redirection, ces données ne « suivent » pas, elles demeurent dans le script PHP d'où est parti la redirection. Il nous faut donc trouver un moyen de les envoyer dans le script « pagecible.php ».

Heureusement, à chaque problème, il y a une solution 😊.

ATTENTION : parmi les solutions, il y en a de très mauvaises. Il se trouve que l'on raconte beaucoup d'âneries sur internet... et même dans certains forums qui ont pignon sur rue. En l'occurrence, la solution préconisée par certains développeurs, consistant à envoyer le contenu de \$_POST sous forme de paramètres dans la redirection, c'est-à-dire concaténés derrière « pagecible.php » à la manière d'une requête HTTP de type GET, est une énorme... bêtise 😞 (j'avais pensé à un autre mot, mais je me suis censuré).

Vous avez peut être du mal à comprendre de quoi je parle, mais je ne veux pas vous montrer d'exemple de code, car un lecteur pressé pourrait croire que c'est la bonne solution. Et comme en plus, ça me hérissé le poil, je ne vous montrerai pas ça.

La bonne solution consiste à faire transiter les informations au moyen d'un tableau PHP qui s'appelle `$_SESSION`. Pour pouvoir utiliser ce tableau, nous devons impérativement démarrer la gestion de session via la fonction PHP `session_start()`. Cette fonction doit être placée au tout début du script, de préférence à la toute première ligne.

A partir du moment où le gestionnaire de session est démarré, nous pouvons alimenter le tableau `$_SESSION` comme bon nous semble. Pour pouvoir récupérer les données de `$_SESSION` dans le script « `pagecible.php` », nous devrons là aussi effectuer un démarrage du gestionnaire de session, avec la fonction `session_start()`.

Je vous propose ci-dessous une nouvelle version de notre formulaire, avec les modifications suivantes :

- Activation du gestionnaire de session
- Stockage des données du formulaire dans un tableau PHP « maison » que j'ai appelé « `gooddata` », puis copie de ce tableau dans `$_SESSION` juste avant de procéder à la redirection

Voici le code source :

```
<?php
// démarrage du gestionnaire de session
session_start();
?><!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire (2)</title>
    <link rel="stylesheet" href="pageform.css">
</head>
<body>
<?php
    // Initialisation du tableau des erreurs
    $erreurs = [];
    $gooddata = [];

    // Boucle pour analyser si des erreurs sont présentes
    foreach($_POST as $keypost=>$valpost) {
        if ($keypost != 'valid') {
            $valpost = trim($valpost);
            if ($valpost == '') {
                $erreurs [$keypost] = "Champ $keypost : zone obligatoire";
            } else {
                $gooddata [$keypost] = $valpost;
            }
        }
    }

    // si le tableau $gooddata contient des données et qu'il n'y a pas d'erreurs
    if (count($gooddata)>0 && count($erreurs)==0) {
        // copie des bonnes données dans $_SESSION
        $_SESSION = $gooddata;
        // redirection vers "pagecible.php"
        header('location: pagecible.php');
    }
}
```

```

        exit; // on stoppe l'exécution du script courant
    }

    if (count($erreurs)>0) {
        echo '<fieldset style="background-color: bisque; border-radius: 5px;">' .PHP_EOL;
        echo '<legend>Liste des erreurs</legend>' .PHP_EOL;
        echo '<ul>' .PHP_EOL;

        foreach($erreurs as $valerr) {
            echo "<li>$valerr</li>" .PHP_EOL;
        }
        echo '</ul>' .PHP_EOL;
        echo '</fieldset>' .PHP_EOL;
    }

    /**
     * Renvoie en sortie la valeur d'un champ de saisie
     * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
     *   'post' (valeur par défaut)
     *   'get'
     * @param $field
     * @param string $method
     * @return string
     */
    function get_form_input_value($field, $method='post') {
        // suppression des blancs parasites et forçage des valeurs en minuscule
        $field = strtolower(trim($field));
        $method = strtolower(trim($method));

        if ($method == 'get') {
            $value = isset($_GET[$field]) ? trim($_GET[$field]): '';
        } else {
            $value = isset($_POST[$field]) ? trim($_POST[$field]): '';
        }
        return $value;
    }

    /**
     * Renvoie en sortie la valeur "selected" pour les champs de type SELECT
     * si la valeur du champ considéré est égale au second paramètre
     * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
     *   'post' (valeur par défaut)
     *   'get'
     * @param $field
     * @param $option_key
     * @param string $method
     * @return string
     */
    function get_form_select_selected($field, $option_key, $method='post') {
        // suppression des blancs parasites et forçage des valeurs en minuscule
        $field = strtolower(trim($field));
        $method = strtolower(trim($method));
        if ($method == 'get') {
            $value = isset($_GET[$field]) ? trim($_GET[$field]): '';
        } else {
            $value = isset($_POST[$field]) ? trim($_POST[$field]): '';
        }
        if ($value == $option_key) {
            return 'selected';
        }
    }
}

```

```

        }
        return '';
    }
?>
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                   placeholder="saisissez un critère de recherche"
                   value=<?php echo get_form_input_value('search'); ?>">
        />
    </p>
    <p>
        <label for="number_field">Exemple de champ "number"</label>
        <input type="number" name="number" id="number_field"
               value=<?php echo get_form_input_value('number'); ?>">
        />
    </p>
    <p>
        <label for="select_field">Exemple de champ "select"</label>
        <select name="myselect" id="select_field">
            <option value="">Choisissez une valeur</option>
            <?php
                foreach(array('A', 'B', 'C') as $viselect) {
                    echo "<option value=\"{$viselect}\">\n".
                        . get_form_select_selected('myselect', $viselect)
                        . ">valeur {$viselect}</option>" . PHP_EOL;
                };
            ?>
        </select>
    </p>
    </fieldset>
    <input type="submit" name="valid" value="Valider" />
</form>
</body>
</html>
```

Dans le script « pagecible.php », il y a aussi quelques modifications mineures :

- Démarrage du gestionnaire de session au tout début du script
- Remplacement de `$_POST` par `$_SESSION` dans la boucle « `foreach` ».

```
<?php
// démarrage du gestionnaire de session
session_start();

echo '<br>Contenu de $_SESSION affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_SESSION as $postkey=>$postval) {
    echo "<li>{$postkey} => {$postval} </li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Refaites un test complet, relisez aussi la liste des 3 exigences que nous avions fixées au début de ce chapitre, vous allez voir que tout fonctionne.

Formulaire en vrac –

Exemple de champ "search"

Exemple de champ "number"

Exemple de champ "select" ▾

Contenu de `$_SESSION` affiché avec une boucle foreach

- search => test
- number => 4
- myselect => B

C'était un gros chapitre, pas vrai ?

Nous avons découvert ici beaucoup de choses :

- Comment gérer un formulaire et boucler tant qu'il contient des erreurs, sans perdre les données déjà saisies par l'utilisateur
- Comment afficher les erreurs
- Comment gérer une redirection vers un autre script – via la fonction `header()` - sans perdre les données saisies par l'utilisateur (grâce au gestionnaire de session)
- Nous avons découvert le gestionnaire de session, avec la fonction `session_start()` et le tableau associatif `$_SESSION`

Prenez le temps de bien relire ce chapitre, de bien retester les différents exemples, de bien vous approprier les techniques que nous venons d'étudier. Vous pouvez le faire au travers d'un formulaire différent si vous le jugez utile.

Si vous butez sur des problèmes relatifs à l'utilisation de `$_POST`, rappelez-vous que ce n'est rien d'autre qu'un tableau associatif, et si cela ne vous rassure pas, relisez les chapitres 4.2.3 et 4.4.2 qui parlent respectivement de tableau et de `$_POST`.

Et n'oubliez pas de faire une pause, je pense que vous l'avez bien méritée 😊.

Quelques remarques complémentaires s'imposent :

- Le code du script « pageform.php » est un peu lourd. Sa lisibilité pourrait être grandement améliorée si certains parties du code étaient déportées dans des fonctions, et placées dans des script distincts importés via la fonction PHP require().
- L'objectif visé dans ce chapitre, c'est la découverte et la compréhension des mécanismes en jeu dans les échanges entre client (navigateur) et serveur (stack PHP).
- L'optimisation du code PHP est certes souhaitable, mais elle peut être réalisée dans un second temps par l'apprenant, ce qui peut constituer un excellent exercice en soi
- Dans le chapitre 5.6 en annexe, vous trouverez une version légèrement différente des scripts « pageform.php » et « pagecible.php ». Dans cette version, le champ de type « select » a été modifié, avec l'ajout de l'attribut « multiple ». Vous verrez dans le chapitre en annexe, que cette modification en apparence mineure a un fort impact sur le code PHP.

4.4.4 Un formulaire robuste et sécurisé

Ce chapitre est une très courte introduction aux problématiques de sécurité liées à la gestion des formulaires.

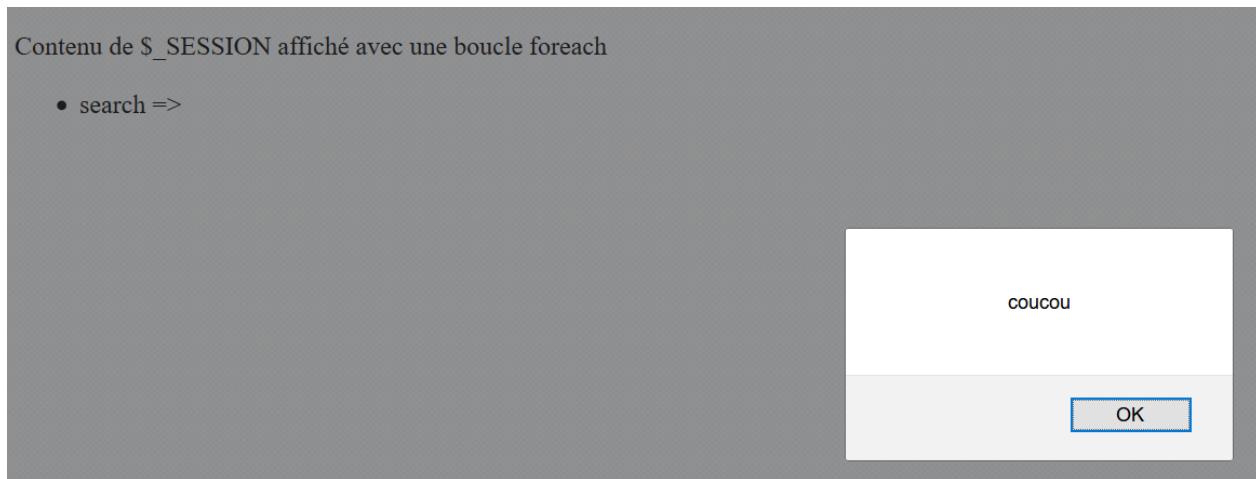
C'est souvent un sujet qui est abordé beaucoup trop tard, quand les développeurs ont pris de mauvaises habitudes. Alors, je préfère aborder le sujet, au moins partiellement, dès maintenant.

Je vous propose de commencer par un test.

Sur la version finale du formulaire du chapitre précédent, saisissez dans le premier champ de saisie (celui qui est de type « text »), le code suivant :

```
<script>alert('coucou');</script>
```

En arrivant sur la page affichée par le script « pagecible.php », on a la mauvaise surprise de voir notre code Javascript s'exécuter, simplement parce que nous avons fait un « echo » du contenu du champ « search » :



Nous venons de mettre le doigt dans un monde bizarre, celui du XSS... du quoi ?

XSS est l'acronyme déformé de « Cross Site Scripting ». En toute logique, on devrait dire CSS, mais l'acronyme CSS étant déjà pris pour autre chose, l'acronyme XSS a finalement été retenu pour éviter toute confusion.

Je vous propose de lire la définition Wikipédia :

Le cross-site scripting (abrégé XSS), est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, permettant ainsi de provoquer des actions sur les navigateurs web visitant la page. Les possibilités des XSS sont très larges puisque l'attaquant peut utiliser tous les langages pris en charge par le navigateur (JavaScript, Java, Flash...) et de nouvelles possibilités sont régulièrement découvertes notamment avec l'arrivée de nouvelles

technologies comme HTML5. Il est par exemple possible de rediriger vers un autre site pour de l'hameçonnage ou encore de voler la session en récupérant les cookies.

Source Wikipédia : https://fr.wikipedia.org/wiki/Cross-site_scripting

Un peu plus bas, dans le même article, il y a un paragraphe qui nous intéresse car il explique comment se protéger contre les attaques XSS, quand on travaille avec le langage PHP. Je le cite :

- utiliser la fonction `htmlspecialchars()` qui filtre les '<' et '>' (cf. ci-dessus) ;
- utiliser la fonction `htmlentities()` qui est identique à `htmlspecialchars()` sauf qu'elle filtre tous les caractères équivalents au codage HTML ou Javascript.
- utiliser `strip_tags()` qui supprime les balises.

Utiliser la fonction `htmlentities()` pour afficher dans une page HTML des données provenant de PHP est une bonne pratique que je recommande, nous allons en reparler dans un instant.

Je vous avoue que j'ai un faible pour la fonction `strip_tags()`, qui a pour effet de dégager les balises HTML qui pourraient être contenues dans des champs de saisie. Dans le cas de notre tentative d'attaque précédente, cela donne ceci :

Avant filtrage via <code>strip_tags()</code>	Après filtrage
<code><script>alert('coucou');</script></code>	<code>alert('coucou');</code>

Débarassé de la balise « script », le contenu du champ de saisie devient inoffensif.

L'imagination des pirates informatiques est sans limites quand il s'agit de mettre un site en difficulté, aussi si on peut leur compliquer la tâche, il ne faut pas s'en priver. Dans notre cas, nous pouvons placer la fonction `strip_tags()` à au moins deux niveaux :

- dans la boucle d'analyse de `$_POST`, lors du chargement des tableaux `$erreurs` et `$gooddata` :

```
// Boucle pour analyser si des erreurs sont présentes
foreach($_POST as $keypost=>$valpost) {
    if ($keypost != 'valid') {
        $valpost = strip_tags($valpost);
        $valpost = trim($valpost);
        if ($valpost == '') {
            $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
        } else {
            $gooddata[$keypost] = $valpost;
        }
    }
}
```

- dans la fonction `get_form_input_value()`, juste avant le « return » :

```
function get_form_input_value($field, $method='post') {
    // suppression des blancs parasites et forçage des valeurs en minuscule
    $field = strtolower(trim($field));
    $method = strtolower(trim($method));

    if ($method == 'get' ) {
        $value = isset($_GET[$field]) ? $_GET[$field] : '';
    } else {
        $value = isset($_POST[$field]) ? $_POST[$field] : '';
    }
    $value = trim(strip_tags($value));
    return $value;
}
```

Et puisque l'utilisation de la fonction `htmlentities()` constitue une bonne pratique, nous allons la mettre en œuvre, mais je souhaiterais auparavant vous montrer en quoi elle est utile.

Prenons l'exemple de notre champ de saisie « search », son code HTML de base est le suivant :

```
<input type="search" name="search" id="search_field"
placeholder="saisissez un critère de recherche" value="">
```

On voit donc que les valeurs contenues dans l'attribut « value » sont circonscrites par des guillemets. Supposons qu'un utilisateur saisisse un guillemet dans ce même champ, et qu'un autre champ de saisie ait été oublié, le navigateur va boucler sur le formulaire, mais l'attribut « value » de notre champ « search » va rencontrer un problème :

```
<input type="search" name="search" id="search_field"
placeholder="saisissez un critère de recherche" value ">
```

Oups, c'est « pas glop » dirait le Pifou. Il y a manifestement un problème du côté de l'attribut « value ». Si on passe en mode « édition » sur ce même champ, ça sent carrément le roussi :

```
<input type="search" name="search" id="search_field"
placeholder="saisissez un critère de recherche" value="" "=">|
```

En fait, on ne s'en rend pas compte au premier coup d'œil, mais notre formulaire est en train de « partir en sucette », tout ça à cause d'un caractère mal échappé.

Et pourtant, c'est tellement simple de se prémunir contre ce genre d'embrouille, avec la fonction `htmlentities()`.

Si à la fin de la fonction `get_form_input_value()`, juste avant le « return », j'ajoute la ligne ci-dessous :

```
$value = htmlentities($value, ENT_QUOTES, "UTF-8");
```

... dans ce cas, les apostrophes et guillemets sont bien échappés, et tout rentre dans l'ordre :

```
<input type="search" name="search" id="search_field" placeholder="saisissez un critère de recherche" value="">
```

On peut d'ailleurs en profiter pour « blinder » le code du script « pagecible.php », histoire de pouvoir dormir sur nos deux oreilles :

```
<?php
// démarrage du gestionnaire de session
session_start();

echo '<br>Contenu de $_SESSION affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_SESSION as $postkey=>$postval) {
    $postkey = htmlentities($postkey, ENT_QUOTES, "UTF-8");
    $postval = htmlentities($postval, ENT_QUOTES, "UTF-8");
    echo "<li>{$postkey} => {$postval}</li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Si dans la vie de tous les jours, la paranoïa est plutôt un problème, en matière de développement logiciel, et en particulier de développement web, la paranoïa est une véritable qualité. Les fonctions `strip_tags()` et `htmlentities()` sont là pour nous aider, alors il ne faut surtout pas s'en priver.

Nous retrouverons d'ailleurs les fonctions `strip_tags()` et `htmlentities()` dans le chapitre suivant, dans lequel nous allons nous amuser à créer une fonction générique capable de générer pour nous un ou plusieurs champs de formulaire.

Il faut souligner que PHP s'est enrichi au fil du temps de fonctions dédiées au filtrage de données (avec possibilité de validation et de nettoyage). Ces techniques viennent compléter le panel de solutions présentées dans ce chapitre, elles sont décrites en annexe (cf. chapitre 5.5).

Ouf, pour une fois c'était un petit chapitre... mais un chapitre important !

4.4.5 Générateur de formulaire

4.4.5.1 Un générateur de champ « input »

Vous aurez sans doute remarqué, en créant des champs de formulaire dans les chapitres précédents, qu'il y a beaucoup de tâches répétitives dans la création d'un champ de formulaire. En effet, pour chaque champ que nous créons, nous devons mettre bout à bout des balises et des attributs HTML... à la longue cela devient barbant.

Avec toutes les connaissances que vous avez accumulées maintenant, vous devriez être en mesure de créer une fonction PHP capable de générer un champ de formulaire de ce type :

```
<input id="id_testchamp1" class="form-control"
name="testchamp1" value="toto@titi.fr" required=""
placeholder="saisissez une adresse email" type="email">
```

Prenez le temps de faire un petit inventaire des éléments qui composent ce champ de saisie :

- nous avons une balise « input »
- nous avons l'attribut « type », généralement fixé à « text » mais qui peut être différent si nous souhaitons générer des types différents comme le type « email » (dans notre exemple), le type « date », ou autre... Comme vous êtes des pros en HTML, vous savez aussi que, si on oublie de définir cet attribut, ou si on lui attribue une valeur erronée, c'est le type « text » qui sera retenu par le navigateur par défaut
- nous avons l'attribut « class », qui est facultatif, et qui est fixé ici à « form-control » parce que nous souhaitons bénéficier ici du rendu proposé par le framework CSS Bootstrap
- nous avons l'attribut « name », qui est facultatif (si notre formulaire est géré uniquement par du JS). Mais dans notre cas, nous voulons que notre formulaire fonctionne tout à fait normalement, et envoie ses données dans la variable PHP \$_POST, donc en ce qui nous concerne l'attribut « name » est obligatoire
- Nous avons l'attribut « id » qui est facultatif. Il est surtout utile si on souhaite manipuler le DOM, encore qu'il soit possible de s'en passer et d'accéder à cet élément par d'autres moyens (il n'y a pas que la fonction getElementById() dans la vie). L'attribut « id » doit surtout être unique, et il est également utile dans notre cas, si nous voulons lier le champ de formulaire à une balise « label » (via l'attribut « for » de la balise « label »).
- L'attribut « required » est facultatif, je rappelle que s'il est défini il a pour effet de rendre notre champ obligatoire
- L'attribut « placeholder » est facultatif, je rappelle que s'il est défini il fournit une assistance à la saisie, si le champ de saisie est vide.

Bon, vous connaissiez tout ça, nous l'avions étudié dans les cours « HTML5 – Premiers pas », « CSS3 – Premiers pas », « Bootstrap – Premiers pas » et « Javascript – Premiers pas ». Mais je pense que c'était intéressant de rappeler ces éléments ici, et de constater que les connaissances accumulées au fil des semaines vont finalement nous être utiles pour créer un composant capable de faire le boulot à notre place.

Pour parvenir à écrire un composant de ce type, quand c'est la première fois qu'on en écrit un, on tâtonne généralement beaucoup avant d'arriver à une version qui fonctionne, mais c'est un exercice très intéressant à réaliser. On apprend généralement beaucoup en faisant ce genre de travail. Si vous avez le temps, je vous encourage à tenter l'exercice, avant de lire la solution que je vous propose dans la suite de ce chapitre.

La version que je vous propose ici est assez proche d'une version que j'avais réalisée avec des élèves, et elle n'est pas très éloignée d'un composant que j'ai utilisé sur certains projets.

```
/**
 * Fonction de génération d'une balise "input"
 * @param $type_champ
 * @param $nom_champ
 * @param string $valeur_champ
 * @param bool $required
 * @param string $placeholder
 * @return string
 */
function gen_form_input($type_champ, $nom_champ, $valeur_champ = '',
                       $required = false, $placeholder = '') {
    if ($required === true) {
        $attribute_required = 'required';
    } else {
        $attribute_required = '';
    }
    $placeholder = trim(strip_tags($placeholder));
    if ($placeholder != '') {
        $placeholder = htmlentities($placeholder, ENT_QUOTES, 'UTF-8');
        $attribute_placeholder = 'placeholder="' . $placeholder . '"';
    } else {
        $attribute_placeholder = '';
    }
    if ($type_champ != 'hidden') {
        $attribute_class = 'class="form-control" ';
    } else {
        $attribute_class = '';
    }
    $nom_champ = trim(strip_tags($nom_champ));
    $valeur_champ = trim(strip_tags($valeur_champ));
    $valeur_champ = htmlentities($valeur_champ, ENT_QUOTES, 'UTF-8');
    return '<input type="'. $type_champ . '"'
        . 'name="'. $nom_champ . '"'
        . 'id="'. $nom_champ . '"'
        . 'value="'. $valeur_champ . '"'
        . $attribute_class. $attribute_required . ''
        . $attribute_placeholder . '/>' . PHP_EOL;
}
```

Il faut souligner aussi qu'il n'y a pas une seule manière de traiter ce sujet, on peut trouver différentes solutions, plus ou moins élégantes, plus ou moins efficaces.

```
// Quelques tests :
echo gen_form_input('email', 'testchamp1', 'toto@titi.fr', true, 'saisissez une adresse email');
echo '<br>';
echo gen_form_input('text', 'testchamp2', '123', true, 'saisissez quelque chose');
echo '<br>';
echo gen_form_input('date', 'testchamp3', '', false, '');
echo '<br>';
echo gen_form_input('date', 'testchamp4');
echo '<br>';
```

Voici le code HTML généré par les 4 tests ci-dessus :

```
<input id="id_testchamp1" class="form-control"
name="testchamp1" value="toto@titi.fr" required=""
placeholder="saisissez une adresse email" type="email">
<br>
<input id="id_testchamp2" class="form-control"
name="testchamp2" value="123" required=""
placeholder="saisissez quelque chose" type="text">
<br>
<input id="id_testchamp3" class="form-control"
name="testchamp3" value="" type="date">
<br>
<input id="id_testchamp4" class="form-control"
name="testchamp4" value="" type="date">
```

Il est important de vérifier la qualité du code HTML produit, quand on crée un composant de ce type... et de multiplier les tests, bien au-delà du petit jeu d'essai ci-dessus.

J'ai mis peu de commentaires dans le code, à ce stade vous devez être en mesure de le comprendre, car je n'ai utilisé que des choses vues dans les précédents chapitres.

Quelques remarques s'imposent :

- J'ai usé et même abusé des fonctions `strip_tags()` et `htmlentities()`, en les appliquant au contenu des paramètres `$nom_champ`, `$valeur_champ` et `$placeholder`. C'est totalement volontaire, je vous rappelle que je suis parano 😊. Mon objectif ici, c'est de créer un composant le plus robuste possible.
 - o Comme je ne sais pas comment il va être utilisé, ni si les informations qui vont lui être transmises en paramètre sont correctement filtrées, je préfère leur appliquer la fonction `strip_tags()`.
 - o Comme je ne sais pas si les caractères tels que les apostrophes et surtout les guillemets – susceptibles de mettre « la grouille » dans mon code HTML – sont correctement échappés dans les paramètres `$valeur_champ` et `$placeholder`, je prends les devants en appliquant la fonction `htmlentities()`.

Une remarque plus générale par rapport au composant que nous venons de créer :

Quand on est un développeur débutant et qu'on s'attelle pour la première fois à un composant de ce type, on a souvent tendance à mettre des « echo » dans ce genre de composant. L'idée sous-jacente, c'est que le composant est dédié à la génération de code HTML, donc si on l'appelle, autant qu'il génère le code HTML par un « echo » plutôt qu'en renvoyant ce code HTML via le mot clé « return ».

Il s'agit d'une erreur courante que j'ai rencontrée à plusieurs reprises, il n'y a pas de honte à avoir 😊.

Mais il s'agit quand même d'une mauvaise idée, car dans la pratique on souhaite le plus souvent décorréler la génération du code HTML (généré par le composant), de son envoi vers le buffer HTML.

Le fait de renvoyer le code HTML via le mot clé « return » nous permet de préparer ce code à l'avance, de le stocker, et de l'utiliser au moment opportun.

Il serait intéressant à ce stade de réfléchir à un composant, qui ferait fonctionnement la même chose, mais qui commencerait par générer les différents attributs dans un tableau PHP, pour ensuite exploiter ce tableau en fin de parcours, pour générer le code HTML final. Je vous laisse y réfléchir.

4.4.5.2 Une première approche de la POO

La POO, ou « Programmation Orientée Objet ».

Le composant que nous avons créé au chapitre précédent pourrait être le premier d'une série de composants dédiés à la génération de formulaires. En effet, il existe de nombreux types de champ de saisie, avec les listes déroulantes, les boutons-radios, les cases à cocher, les labels, les champs de type « textarea », etc...

Donc il semble logique de vouloir créer un panel de composants plus large que notre simple fonction `gen_form_input()`.

J'allais l'oublier, mais nous avons aussi des balises « input » de type « hidden » et « password », en plus des types « text » et des nouveaux types du HTML5 (« date », « search », « email », etc....).

En nous appuyant sur notre fonction `gen_form_input()`, nous pouvons créer un jeu de fonctions plus spécialisées, comme par exemple :

```
function gen_input_text($nom_champ, $valeur_champ = '',
    $required = false, $placeholder = '') {
    return gen_form_input('text', $nom_champ, $valeur_champ,
        $required, $placeholder);
}

function gen_input_hidden($nom_champ, $valeur_champ) {
    return gen_form_input('hidden', $nom_champ, $valeur_champ);
}

function gen_input_password($nom_champ, $valeur_champ = '',
    $required = false, $placeholder = '') {
    return gen_form_input('password', $nom_champ, $valeur_champ,
        $required, $placeholder);
}
```

Et ce n'est qu'un petit panel de ce que nous pouvons créer dans ce domaine.

Mais du coup, toutes ces petites fonctions autonomes... ça va vite devenir le bazar !

Il serait intéressant de les regrouper... par exemple dans une classe. En POO, une classe, c'est en quelque sorte le mode d'emploi – ou plus exactement le mode de fonctionnement - d'un objet. Une classe PHP se définit par le mot clé « class », suivi d'un nom, et d'accolades à l'intérieur desquelles nous pouvons placer nos différentes fonctions, qui pour le coup s'appellent des « méthodes » (en jargon POO).

Dans l'exemple ci-dessous, j'ai créé une classe que j'ai appelée `GenForm`, dans laquelle j'ai placé mes différentes fonctions (méthodes), j'ai retiré tout le gras, pour que l'on puisse se concentrer sur la structure de la classe :

```

class GenForm {
    public function gen_form_input($type_champ, $nom_champ, $valeur_champ = '',
                                    $required = false, $placeholder = '') {
        // code retiré pour alléger l'exemple
    }
    public function gen_input_text($nom_champ, $valeur_champ = '',
                                   $required = false, $placeholder = '') {
        return $this->gen_form_input('text', $nom_champ, $valeur_champ,
                                      $required, $placeholder);
    }
    public function gen_input_hidden($nom_champ, $valeur_champ) {
        return $this->gen_form_input('hidden', $nom_champ, $valeur_champ);
    }
    public function gen_input_password($nom_champ, $valeur_champ = '',
                                       $required = false, $placeholder = '') {
        return $this->gen_form_input('password', $nom_champ, $valeur_champ,
                                     $required, $placeholder);
    }
}

```

Vous voyez que l'on retrouve nos 4 fonctions, elles portent d'ailleurs toujours le nom de « function » malgré le fait qu'il s'agit de méthodes d'une classe. C'est une bizarrerie de PHP... no comment.

Quelques détails importants à noter :

- la présence du mot clé « public » devant le mot clé « function ». Ce mot clé indique que les fonctions sont visibles depuis l'extérieur de l'objet, on dit qu'elles sont « publiques ». Nous verrons dans un instant comment ça s'utilise
- la présence du mot clé « \$this » suivi d'une flèche, avant chaque appel de la méthode « gen_form_input ». Ce mot clé « \$this » indique que nous faisons appel à une méthode qui appartient à l'objet en cours. Par exemple, la méthode gen_input_text() appartient à un objet et utilise la méthode gen_form_input() qui appartient au même objet.

J'ai oublié de préciser, que pour que notre objet fonctionne, nous devons l'instancier de cette manière :

```
$objForm = new GenForm();
```

Eh oui, sans cette action d'instanciation, la classe GenForm est inutilisable. Je rappelle que ce n'est qu'une classe, pas un objet. Une classe c'est la notice descriptive du fonctionnement de l'objet, ce n'est pas l'objet lui-même. L'objet, c'est dans notre exemple la variable \$objForm.

Voici quelques exemples d'utilisation de l'objet \$objForm, et surtout de ses méthodes publiques :

```
echo $objForm->gen_form_input('email', 'testchamp1', 'toto@titi.fr',
                                true, 'saisissez une adresse email');
echo '<br>';
echo $objForm->gen_input_text('testchamp2', '123', false,
                                'saisissez quelque chose');
echo '<br>';
echo $objForm->gen_input_hidden('testhidden', 'champ caché');
echo '<br>';
echo $objForm->gen_input_password('testpassword');
echo '<br>';
```

Le meilleur moyen de s'assurer du bon fonctionnement de ces méthodes, c'est d'observer le code HTML généré (via l'inspecteur d'élément du navigateur) :

```
<input type="email" name="testchamp1" id="id_testchamp1" value="toto@titi.fr"
       class="form-control" required placeholder="saisissez une adresse email">
<br>
<input type="text" name="testchamp2" id="id_testchamp2" value="123" class="form-
control" required placeholder="saisissez quelque chose">
<br>
<input type="date" name="testchamp3" id="id_testchamp3" value class="form-control">
<br>
<input type="date" name="testchamp4" id="id_testchamp4" value class="form-control">
<br>
<input type="text" name="testchampx" id="id_testchampx" value class="form-control"
       required placeholder="saisissez une \\"adresse email">
<input type="text" name="testchampy" id="id_testchampy" value="toto@titi.fr" class=
"form-control" required>
```

Vous trouverez page suivante une version légèrement modifiée de la classe GenForm, dans laquelle j'ai souhaité disposer de la possibilité de générer des champs de saisie dotés de l'attribut « disabled ». J'en aurai besoin dans un autre chapitre de ce support, le chapitre relatif au CRUD.

```

/**
 * Class GenForm
 * Regroupe des méthodes dédiées à la génération de champs de saisie :
 */
class GenForm {

    /**
     * Fonction de génération d'une balise "input"
     * @param $type_champ
     * @param $nom_champ
     * @param string $valeur_champ
     * @param bool $required
     * @param string $placeholder
     * @param bool $disabled
     * @return string
     */
    public function gen_form_input($type_champ, $nom_champ, $valeur_champ = '',
                                   $required = false, $placeholder = '', $disabled=false)
    {

        if ($required === true) {
            $attribute_required = 'required';
        } else {
            $attribute_required = '';
        }
        if ($disabled === true) {
            $attribute_disabled = 'disabled';
        } else {
            $attribute_disabled = '';
        }
        $placeholder = trim(strip_tags($placeholder));
        if ($placeholder != '') {
            $placeholder = htmlentities($placeholder, ENT_QUOTES, 'UTF-8');
            $attribute_placeholder = 'placeholder="' . $placeholder . '"';
        } else {
            $attribute_placeholder = '';
        }
        if ($type_champ != 'hidden') {
            $attribute_class = 'class="form-control" ';
        } else {
            $attribute_class = '';
        }
        $nom_champ = trim(strip_tags($nom_champ));
        $valeur_champ = trim(strip_tags($valeur_champ));
        $valeur_champ = htmlentities($valeur_champ, ENT_QUOTES, 'UTF-8');
        return '<input type="' . $type_champ . '"'
            . 'name="' . $nom_champ . '"'
            . 'id="id_' . $nom_champ . '"'
            . 'value="' . $valeur_champ . '"'
            . $attribute_class
            . $attribute_required . ''
            . $attribute_disabled . ''
            . $attribute_placeholder . '/>' . PHP_EOL;
    }

    public function gen_input_text($nom_champ, $valeur_champ = '',
                                  $required = false, $placeholder = '', $disabled=false)
    {
        return $this->gen_form_input('text', $nom_champ, $valeur_champ,
                                     $required, $placeholder, $disabled);
    }
}

```

```

    }

    public function gen_input_hidden($nom_champ, $valeur_champ)
    {
        return $this->gen_form_input('hidden', $nom_champ, $valeur_champ);
    }

    public function gen_input_password($nom_champ, $valeur_champ = '',
                                       $required = false, $placeholder = '')
    {
        return $this->gen_form_input('password', $nom_champ, $valeur_champ,
                                      $required, $placeholder);
    }

}

// quelques lignes de test
$objForm = new GenForm();

echo $objForm->gen_form_input('email', 'testchamp1', 'toto@titi.fr', true,
                               'saisissez une adresse email');
echo '<br>';
echo $objForm->gen_input_text('testchamp2', '123', false, 'saisissez quelque chose');
echo '<br>';
echo $objForm->gen_input_hidden('testhidden', 'champ caché');
echo '<br>';
echo $objForm->gen_input_password('testpassword');
echo '<br>';

```

Il y a encore plein de choses à voir concernant la POO, ce chapitre n'est qu'une courte introduction. Nous étudierons le sujet plus en détail dans un prochain chapitre, mais vous avez déjà acquis quelques connaissances essentielles sur...

- la notion de classe,
- la notion d'objet objet (et la manière dont on instancie un objet),
- la notion de méthode, et en particulier de méthode publique
- la manière dont une méthode peut utiliser une autre méthode du même objet (via le mot clé « `$this` »)

Avec le recul que j'ai acquis au fur et à mesure des formations, je dirais que créer une classe contenant des méthodes dédiées à la génération de formulaire, c'est un excellent outil pédagogique pour appréhender la POO et commencer à s'approprier le sujet.

Mais il est temps de s'attaquer à un autre sujet, car je sais que vous avez hâte de découvrir de quelle manière PHP peut fonctionner avec une base SQL.

4.5 Les objets en PHP

Ce chapitre est une introduction à la POO (Programmation Orientée Objet) en PHP. Nous n'allons pas couvrir tous les aspects de la POO (sinon il faudrait au moins tripler la taille de ce support), mais nous allons nous efforcer les points essentiels, qui vous permettront d'appréhender le fonctionnement des frameworks, qui pour la plupart utilisent massivement des classes et des objets.

La documentation officielle disponible sur [php.net](http://www.php.net/manual/fr/language.oop5.php) propose plusieurs chapitres très complets sur la POO :

<http://www.php.net/manual/fr/language.oop5.php>

4.5.1 Classe « découverte »

Une classe est un bloc de code composé de variables (appelées "propriétés" dans le jargon objet) et de fonctions (appelées "méthodes" dans le jargon objet).

Une classe peut donc être vue comme une manière élégante de regrouper sous un même toit des données, ainsi que les fonctionnalités associées à ces données.

La raison d'être d'une classe, c'est d'être utilisée pour instancier des objets (ce n'est pas tout à fait vrai, car il y a des exceptions, mais pour l'instant disons que c'est vrai 😊). Ces objets seront autant d'entités autonomes, reposant sur le même squelette que constitue la classe, mais hébergeant chacune un jeu de données spécifiques.

Voici un premier exemple de classe qui ne contient que des propriétés :

```
class Album {  
    public $artiste;  
    public $titre;  
    public $pistes;  
    public $support;  
    public $format;  
    public $disk_id;  
}
```

Vous l'aurez compris, il s'agit d'une classe servant à décrire des albums de musique. Je vous propose de créer une première instance de cette classe, donc un objet, et cet objet sera stocké dans une variable portant le joli nom de \$album1.

Voici donc une première instance de la classe Album :

```
$album1 = new Album();
$album1->artiste = 'Her';
$album1->titre = 'Her Tape #1';
$album1->support = 'CD';
$album1->format = 'EP';
$album1->disk_id = 1;
$album1->pistes = [];
$album1->pistes[] = 'Intro';
$album1->pistes[] = 'Quite Like';
$album1->pistes[] = 'Five Minutes';
$album1->pistes[] = 'Interlude';
$album1->pistes[] = 'Her';
$album1->pistes[] = 'Union';
```

Dès que notre instance est créée (et alimentée en données), nous pouvons l'utiliser pour afficher des données :

```
echo 'Titre : ' . $album1->titre . '<br>';
echo 'Artiste : ' . $album1->artiste . '<br>';
echo 'Titres :<br>';
echo '<ol>';
foreach($album1->pistes as $piste) {
    echo '<li>' . $piste . '</li>';
}
echo '</ol>';
```

Nous pouvons dès lors créer un tableau PHP stockant plusieurs albums. Bon, j'en crée juste deux, et comme je suis un peu paresseux, je ne déclare que les titres et les artistes :

```
$liste_albums = []

$liste_albums[0] = new Album();
$liste_albums[0]->titre = 'Her Tape #1';
$liste_albums[0]->artiste = 'Her';

$liste_albums[1] = new Album();
$liste_albums[1]->titre = 'Convergence';
$liste_albums[1]->artiste = 'Malia / Boris Blank';

$liste_albums[2] = new Album();
$liste_albums[2]->titre = 'Wandering in Dust';
$liste_albums[2]->artiste = 'Doedelzak';
```

Bon, je ne sais pas pour vous, mais en ce qui me concerne, je ne trouve ça ni pratique, ni élégant. Heureusement, il y a la méthode « Constructeur ».

Je vous propose de revoir notre copie, avec une classe Album Version 2 :

```
class AlbumV2 {
    public $artiste;
    public $titre;
    public $pistes;
    public $support;
    public $format;
    public $disk_id;

    public function __construct($artiste, $titre, $support='', $format='',
        $disk_id=0, $pistes=array())
    {
        $this->artiste = $artiste;
        $this->titre = $titre;
        $this->support = $support;
        $this->format = $format;
        $this->id = $disk_id;
        $this->pistes = $pistes;
    }
}
```

La méthode `__construct()` fait partie de ce qu'on appelle les « méthodes magiques » (il en existe d'autres que nous verrons plus tard). C'est le constructeur de la classe, on peut lui passer tout un tas de paramètres, et les stocker dans les propriétés de la classe au moyen du mot clé « `this` » suivi d'une petite flèche :

```
$this->artiste = $artiste;
```

On retrouve dans les méthodes d'une classe la possibilité de rendre des paramètres optionnels, en leur attribuant une valeur par défaut (c'est le cas dans notre exemple de la variable `$support`, et de celles qui suivent).

Tiens, c'est marrant, la création des objets devient tout de suite plus agréable :

```
$liste_albums = [];
$liste_albums[] = new AlbumV2('Her', 'Her Tape #1');
$liste_albums[] = new AlbumV2('Malia / Boris Blank', 'Convergence');
$liste_albums[] = new AlbumV2('Doedelzak', 'Wandering in Dust');
```

Bien évidemment, si je veux que l'album « `Her` » retrouve toutes ses pistes, je dois transmettre le tableau correspondant au constructeur lors de l'instanciation :

```
$pistes = [];
$pistes[] = 'Intro';
$pistes[] = 'Quite Like';
$pistes[] = 'Five Minutes';
$pistes[] = 'Interlude';
$pistes[] = 'Her';
$pistes[] = 'Union';
$liste_albums[] = new AlbumV2('Her', 'Her Tape #1', 'CD', 'EP', 1, $pistes);
```

Pour l'instant, toutes les propriétés et méthodes de la classe Album sont définies avec une visibilité publique. Cela signifie qu'elles sont accessibles depuis l'extérieur de l'objet, ce qui nous permet d'écrire ceci :

```
$album_prefered = new AlbumV2('Her', 'Her Tape #1');  
$album_prefered->titre = 'Herk' ;
```

Dans des projets professionnels, vous serez peut être amené à renforcer la fiabilité de certains composants en définissant des méthodes et propriétés publiques, mais aussi des méthodes et propriétés privées (private) ou protégées (protected).

En résumé, nous avons 3 types de visibilité, qui sont :

- public : signifie que la variable ou la méthode est entièrement accessible par tout code situé à l'extérieur de la classe considérée.
- private : signifie que la variable ou méthode n'est visible qu'à l'intérieur de l'objet lui-même.
- protected : se situe à mi-chemin des deux autres options. Une variable ou une méthode déclarée en tant que telle peut être consultée à l'intérieur de l'objet lui-même, ou à partir de tout enfant de cet objet (cas de figure que l'on rencontre quand on utilise la notion d'héritage). L'élément (propriété ou méthode) ne peut pas être accessible de l'extérieur.

Dans le cas de notre classe AlbumV2, si nous décidons de passer nos propriétés en mode « private », cela signifie que nous n'avons plus le droit d'écrire ceci :

```
$album_prefered = new AlbumV2('Her', 'Her Tape #1');  
$album_prefered->titre = 'Herk' ;
```

... sinon, nous nous exposons à une erreur fatale, que voici :

```
Fatal error: Uncaught Error: Cannot access private property AlbumV3::$titre in  
D:\Tools\xampp\htdocs\crud2\class_intro\exemple01.php:121 Stack trace: #0 {main} thrown in  
D:\Tools\xampp\htdocs\crud2\class_intro\exemple01.php on line 121
```

Si nous voulons dans ce contexte conserver la possibilité de modifier une propriété après l'instanciation de l'objet, alors nous devons ajouter un jeu de méthodes publiques de type « set » et « get », qu'on désigne généralement sous les termes de « setters » et de « getters ».

Pour voir ce que ça donne, je vous propose une version 3 de notre classe Album :

```
class AlbumV3 {  
    private $artiste;  
    private $titre;  
    private $pistes;  
    private $support;  
    private $format;  
    private $disk_id;  
  
    public function __construct($artiste, $titre, $support='', $format='',  
                                $disk_id=0, $pistes=array())  
    {  
        $this->artiste = $artiste;  
        $this->titre = $titre;  
        $this->support = $support;  
        $this->format = $format;  
        $this->disk_id = $disk_id;  
        $this->pistes = $pistes;  
    }  
  
    public function getArtiste() {  
        return $this->artiste;  
    }  
    public function getTitre() {  
        return $this->titre;  
    }  
    public function getsupport() {  
        return $this->support;  
    }  
    public function getFormat() {  
        return $this->format;  
    }  
    public function getDiskId() {  
        return $this->disk_id;  
    }  
    public function getPistes() {  
        return $this->pistes;  
    }  
  
    public function setArtiste($artiste) {  
        $this->artiste = $artiste;  
    }  
    public function setTitre($titre) {  
        $this->titre = $titre;  
    }  
    public function setsupport($support) {  
        $this->support = $support;  
    }  
    public function setFormat($format) {  
        $this->format = $format;  
    }  
    public function setDiskId($disk_id) {  
        $this->disk_id = $disk_id;  
    }  
    public function setPistes($pistes) {  
        $this->pistes = $pistes;  
    }  
}
```

Du coup, on ne peut plus écrire ceci :

```
$album_prefered = new AlbumV3('Her', 'Her Tape #1');
echo $album_prefered->titre = 'Herk' ; // BAD : ça va planter !!
```

... par contre on peut écrire cela :

```
$album_prefered = new AlbumV3('Her', 'Her Tape #1');
echo $album_prefered->setTitre('Herk') ; // GOOD BOY !
```

Bon, c'est bien joli tout ça, mais est-ce qu'on n'est pas en train de couper les poils de c.. en quatre ? Est-ce que ça a réellement du sens d'écrire autant de code pour un si maigre bénéfice ? Je suis d'accord avec vous, en l'état cela ne sert pas à grand-chose, sauf si nous blindons le code de nos méthodes en contrôlant le type des données reçues. Dans le cas de la méthode setTitre(), nous pourrions par exemple écrire ceci :

```
public function setTitre($titre) {
    if (!is_string($titre) ) {
        throw new Exception('Type de donnée erronée');
    } else {
        $titre = trim($titre);
        if (strlen($titre)>80) {
            throw new Exception('Donnée supérieure à 80c');
        } else {
            // Suppression des blancs et forçage en majuscule
            $this->titre = strtoupper($titre);
        }
    }
}
```

Dans cet exemple, j'ai fait en sorte que la méthode :

- déclenche une erreur fatale si on lui transmet autre chose qu'une chaîne de caractères
- déclenche une erreur fatale si on lui transmet une chaîne de longueur supérieure à 80 caractères
- supprime les blancs éventuels devant et derrière la donnée transmise et mettre le titre en majuscules avant de le stocker dans la propriété privée « titre »

Voici un petit jeu d'essai que je vous invite à tester :

```
$album_prefered = new AlbumV3('Her', 'Her Tape #1');
echo $album_prefered->setTitre(array()) ;
//=> Fatal error: Uncaught Exception: Type de donnée erronée

echo $album_prefered->setTitre(str_repeat("*", 80)) ;
//=> longueur OK, ça passe :

echo $album_prefered->setTitre(str_repeat("*", 81)) ;
//=> Fatal error: Uncaught Exception: Donnée supérieure à 80c
```

On peut bien évidemment blinder l'ensemble des « setters » sur ce modèle si on le souhaite.

Ah oui, vous avez remarqué que j'ai utilisé une technique particulière pour déclencher une erreur fatale :

```
if (!is_string($titre) ) {
    throw new Exception('Type de donnée eronnée');
}
```

Est-ce justifié de déclencher une erreur fatale dans un cas de figure comme celui-là ? Pas forcément, mais je voulais vous montrer la technique. Concernant la pertinence, je pense que c'est un sujet dont vous devez discuter avec les membres de votre équipe de développement : quelles sont les erreurs qui doivent déclencher des erreurs graves, quelles sont celles qui peuvent être tolérées, moyennant peut-être l'envoie d'un avertissement dans la log de PHP. Je n'ai pas d'avis tranché pour ma part, la gravité de l'erreur doit s'apprécier selon le contexte, et il faut surtout qu'un consensus se dégage au sein d'une équipe de développement, pour éviter que certains développeurs gèrent le problème d'une manière et d'autres d'une autre.

On notera que dans notre classe AlbumV3, la propriété doit recevoir une valeur de type tableau, et non pas de type chaîne. Cela nous donne donc :

```
public function setPistes($pistes) {
    if (!is_array($pistes) ) {
        throw new Exception('Type de donnée eronné');
    } else {
        $this->pistes = $pistes;
    }
}
```

Mais au fait, si on laisse la méthode Constructeur en l'état, elle n'est pas du tout blindée, contrairement aux méthodes « setters ». Voici une manière élégante et efficace de remédier au problème :

```
public function __construct($artiste, $titre, $support='', $format='',
                           $disk_id=0, $pistes=array())
{
    $this->setArtiste($artiste);
    $this->setTitre($titre);
    $this->setSupport($support);
    $this->setFormat($format);
    $this->setDiskId($disk_id);
    $this->setPistes($pistes);
}
```

Le constructeur se contente ici d'un rôle de « passe-plat » : il reçoit des infos et les transmet aux « setters » qui se chargent de contrôler la validité des données et de les stocker au bon endroit.

Voici un autre exemple, avec la classe AlbumV4. Dans cette nouvelle version, les méthodes setFormat et setSupport sont plus « intelligentes », puisqu'elles contrôlent la validité des infos reçues et n'autorisent que certaines valeurs :

```
class AlbumV4 {
    private $artiste;
    private $titre;
    private $pistes;
    private $support;
    private $format;
    private $disk_id;

    public function __construct($artiste, $titre, $support='CD', $format='LP',
                                $disk_id=0, $pistes=array())
    {
        $this->setArtiste($artiste);
        $this->setTitre($titre);
        $this->setSupport($support);
        $this->setFormat($format);
        $this->setDiskId($disk_id);
        $this->setPistes($pistes);
    }

    public function getArtiste() {
        return $this->artiste;
    }
    public function getTitre() {
        return $this->titre;
    }
    public function getSupport() {
        return $this->support;
    }
    public function getFormat() {
        return $this->format;
    }
    public function getDiskId() {
        return $this->disk_id;
    }
    public function getPistes() {
        return $this->pistes;
    }

    public function setArtiste($value) {
        if (!is_string($value) ) {
            throw new Exception('Type de donnée erronée');
        } else {
            $value = trim($value);
            if (strlen($value)>80) {
                throw new Exception('Donnée supérieure à 80c');
            } else {
                // Suppression des blancs et forçage en majuscule
                $this->artiste = $value;
            }
        }
    }
}
```

```
        }

    }

    public function setTitre($value) {
        if (!is_string($value) ) {
            throw new Exception('Type de donnée éronnée');
        } else {
            $value = trim($value);
            if (strlen($value)>80) {
                throw new Exception('Donnée supérieure à 80c');
            } else {
                // Suppression des blancs et forçage en majuscule
                $this->titre = strtoupper($value);
            }
        }
    }

    public function setSupport($support) {
        $liste = array('CD', 'MP3', 'K7');
        $support = strtoupper($support);
        if (in_array($support, $liste) == false) {
            throw new Exception('Type de support invalide');
        } else {
            $this->support = $support;
        }
    }

    public function setFormat($format) {
        $liste = array('EP', 'LP');
        $format = strtoupper($format);
        if (in_array($format, $liste) == false) {
            throw new Exception('Type de format invalide');
        } else {
            $this->format = $format;
        }
    }

    public function setDiskId($disk_id) {
        $this->disk_id = (int)$disk_id;
    }

    public function setPistes($pistes) {
        if (!is_array($pistes) ) {
            throw new Exception('Type de donnée éronnée');
        } else {
            $this->pistes = $pistes;
        }
    }

}
```

```
// Exemple de code permettant de tester la classe
$test2 = new AlbumV4('Her', str_repeat('*', 80));
echo $test2->getArtiste(). '<br>';
echo $test2->getTitre(). '<br>';
echo $test2->getSupport(). '<br>';
echo $test2->getFormat(). '<br>';
if (count($test2->getPistes())>0) {
    echo count($test2->getPistes()) . ' pistes<br>';
} else {
    echo 'pas de pistes';
}
```

J'espère que cette rapide introduction à la POO vous a permis de comprendre quelques uns des principes fondamentaux des classes et des objets. Dans les chapitres qui suivent, je vais passer - quelquefois brièvement, quelquefois de manière plus approfondie – sur certains points complémentaires qu'il est important de connaître pour se sentir à l'aise avec la POO.

4.5.2 Classe abstraites et méthodes statiques

Dans certains cas, vous pourrez être amenés à créer des classes qui n'ont pas pour vocation à être instanciées. Ce sera notamment le cas si vous souhaitez que ces classes soient utilisées comme socle pour la construction d'autres classes, via le mécanisme d'héritage. Pour expliquer le principe, je vous propose de copier la classe AlbumV4 et de créer une classe AlbumV5, en ajoutant le mot clé « abstract » à cette dernière, comme ceci :

```
abstract class AlbumV5 {
    protected $artiste;
    protected $titre;
    protected $pistes;
    protected $support;
    protected $format;
    protected $disk_id;

    //... je ne mets pas le reste du code, c'est le même que celui de AlbumV4
}
```

Vous remarquerez que j'ai modifié la visibilité des propriétés, qui sont maintenant paramétrées à « protected ». C'est nécessaire car je souhaite que ces propriétés « descendent » dans la (ou les) classe(s) fille(s).

Je crée maintenant une classe JazzAlbum, qui hérite des caractéristiques de la classe AlbumV5. On dit que JazzAlbum est une classe fille de AlbumV5 :

```
class JazzAlbum extends AlbumV3 {

    public function getTitreSexy () {
        return '****' . $this->getTitre() . '****';
    }
}
```

Au travers de l'objet \$jazzalbum – instancié à partir de la classe JazzAlbum – j'ai le droit d'utiliser aussi bien les méthodes héritées de la classe mère (comme getTitre) que celles définies dans la classe JazzAlbum elle-même (en l'occurrence la méthode getTitreSexy) :

```
$jazzalbum = new JazzAlbum('Coltrane', 'Blue Train');
echo $jazzalbum->getTitre(); //=> BLUE TRAIN
echo $jazzalbum->getTitreSexy(); //=> **** BLUE TRAIN ****
```

Par contre, je n'ai pas le droit d'instancier un objet à partir de la classe AlbumV5, car je l'ai déclarée avec le mot clé « abstract ». Vous pouvez essayer, vous obtiendrez une erreur grave.

4.5.3 Astuces d'héritier

Il serait trop long de détailler toutes les subtilités de l'héritage de classes, mais je souhaite quand même vous montrer quelques trucs.

Premier truc, il est possible de réécrire une méthode dans la classe fille. Par exemple, si je considère que pour les albums de Jazz, je souhaite pouvoir enregistrer des titres de 100 caractères maxi au lieu de 80, je peux écrire ceci :

```
class JazzAlbum extends AlbumV4 {

    public function getTitreSexy () {
        return '****' . $this->getTitre() . '****';
    }

    public function setTitre($titre) {
        if (!is_string($titre) ) {
            throw new Exception('Type de donnée erronée');
        } else {
            $titre = trim($titre);
            if (strlen($titre)>100) {
                throw new Exception('Donnée supérieure à 80c');
            } else {
                // Suppression des blancs et forçage en majuscule
                $this->titre = strtoupper($titre);
            }
        }
    }
}
```

Voilà, le code de la méthode setTitre() a été réécrit dans la classe fille, du coup celui de la classe mère est ignoré. On dirait bien que cette classe JazzAlbum a envie de prendre son indépendance 😊.

Un second truc que je souhaite vous montrer, c'est qu'il est possible de réécrire une méthode dans une classe fille tout en conservant le code de la classe mère. Par exemple, dans JazzAlbum, je souhaite que la méthode Constructeur envoie un message dans la log de PHP, chose que ne faisait pas la méthode équivalente de la classe mère. Je peux donc écrire ceci dans la classe JazzAlbum :

```
public function __construct($artiste, $titre, $support='', $format='',
                           $disk_id=0, $pistes=array())
{
    error_log('coucou je suis JazzAlbum et vous, vous êtes qui ?');
    // Appel du constructeur de la classe parente
    parent::__construct($artiste, $titre, $support, $format,
                        $disk_id, $pistes);
}
```

Je vous invite à tester cette version de la classe JazzAlbum, et surtout à vérifier le contenu de la log PHP après exécution.

Voilà, c'était ça l'astuce, avec le mot clé « parent::» vous pouvez appeler des méthodes de la classe mère à l'intérieur de méthodes de la classe fille. Cette technique offre beaucoup de souplesse. Cela va peut être vous sembler déroutant au premier abord, mais avec un peu de pratique, je ne doute pas que vous allez apprécier cette souplesse.

4.5.4 Interfaces

L'utilisation d'interfaces de classes constitue un excellent moyen de renforcer la robustesse d'une application orientée objet. En effet, tout développeur qui implémente une interface doit scrupuleusement respecter les règles d'appel définies dans l'interface. S'il commet la moindre erreur, par exemple en omettant un paramètre dans une méthode statique, l'interpréteur PHP va la détecter et stopper le traitement avec un erreur fatale.

Dans le cas de notre classe AlbumV4, si on suppose que toutes ses classes filles doivent respecter scrupuleusement la structure de ses méthodes publiques, alors on peut écrire l'interface suivante :

```
interface AlbumV4interface {
    public function getArtiste() ;
    public function getTitre() ;
    public function getSupport() ;
    public function getFormat() ;
    public function getDiskId() ;
    public function getPistes() ;
    public function setArtiste($artiste) ;
    public function setTitre($titre) ;
    public function setSupport($support) ;
    public function setFormat($format) ;
    public function setDiskId($disk_id) ;
    public function setPistes($pistes) ;
}
```

Il ne reste plus qu'à ajouter cette interface sur la classe JazzAlbum via le mot clé « implements », et le tour est joué :

```
class JazzAlbum extends AlbumV4 implements AlbumV4interface {
    // code identique à celui du chapitre précédent
}
```

Si je modifie la méthode setTitre dans l'interface, en lui ajoutant un paramètre qui n'existe pas dans la méthode équivalente de la classe JazzAlbum, alors je déclenche une erreur fatale :

Fatal error: Declaration of AlbumV4::setArtiste(\$artiste) must be compatible with
AlbumV4interface::setArtiste(\$artiste, \$toto)

Cette notion d'interface est intéressante, car elle permet de définir une sorte de « contrat ». Les classes filles doivent respecter le contrat qui les lie à cette interface. Si plusieurs développeurs créent des classes filles basées sur la classe AlbumV4, et qu'ils implémentent la même interface,

alors cela garantit que toutes les classes filles sont déclarées de manière homogène. Cela permettra d'éliminer certains types d'erreur.

4.5.5 Méthodes magiques

En termes de méthodes magiques, nous avons vu pour l'instant la méthode `__construct()`. Nous allons voir maintenant un exemple de classe utilisant d'autres méthodes magiques, comme `__set()`, `__get()`, `__isset()` et `__unset()` :

```
// Classe dédiée au stockage de données
class DataHolder {
    // tableau de stockage interne "privé"
    private $data = array();

    // méthode déclenchée automatiquement dès qu'une méthode
    // est appelée qui n'existe pas dans l'objet
    public function __set($name, $value) {
        $this->data[$name] = $value;
    }

    // méthode déclenchée automatiquement dès qu'on demande
    // l'affichage d'une méthode qui n'existe pas dans la classe
    public function __get($name) {
        if (isset($this->data[$name])) { return $this->data[$name]; }
    }

    // méthode permettant d'utiliser la fonction "isset" sur les variables
    // stockées dans le tableau interne
    public function __isset($name) {
        return isset($this->data[$name]);
    }

    // méthode permettant d'utiliser la fonction 'unset' sur les variables
    // stockées dans le tableau interne
    public function __unset($name) {
        if (isset($this->data[$name])) {
            unset($this->data[$name]);
        }
    }
}

$data = new DataHolder();
// les propriétés n'existent pas, ce qui va déclencher l'appel de __set()
$data->titre = 'Le chat du rabbin';
$data->auteur = 'Joann Sfar';

echo "<p>Titre : {$data->titre} | Auteur : {$data->auteur}</p>";
//=> Titre : Le chat du rabbin | Auteur : Joann Sfar
```

4.5.6 Trucs et astuces

Destruction d'un objet :

```
unset ($object) ;
```

Toutes les ressources associées à l'objet sont supprimées. Si la classe considérée contient une méthode magique `__destruct()`, alors cette dernière est appelée automatiquement.

Documentation: <http://php.net/unset> and <http://php.net/destruct>

Création d'une copie d'un objet :

```
$object_copy = clone $object;
```

En PHP 5, une simple affectation crée seulement une référence à un objet. L'ordre "clone" doit donc être utilisé pour créer toute une copie d'un objet. Si une méthode `__clone()` est définie pour la classe considérée, elle sera appelée.

Documentation: <http://php.net/clone>

Comparaison de 2 objets distincts :

```
$isequal = ($obj1 == $obj2) ;
```

L'opérateur de comparaison `==`, vérifie si les deux objets sont de la même classe et si toutes les variables de classe ont la même valeur.

Documentation: <http://php.net/manual/fr/language.oop5.object-comparison.php>

Comparer si deux variables d'objets font référence à la même instance d'un objet:

```
$issameinstance = ($obj1 === $obj2) ;
```

L'opérateur d'identité `===` retourne vrai si les deux variables d'objet font référence à la même instance d'un objet.

Documentation: <http://php.net/manual/fr/language.oop5.object-comparison.php>

Il est possible de convertir facilement un tableau (ou tout autre type de variable) en objet. Cette technique est assez bien expliquée sur php.net, en voici un extrait :

<http://php.net/manual/fr/language.types.object.php>

Si un objet est converti en un objet, il ne sera pas modifié. Si une valeur de n'importe quel type est convertie en un objet, une nouvelle instance de la classe interne stdClass sera créée. Si la valeur est NULL, la nouvelle instance sera vide. Un array se converti en object avec les propriétés nommées au regard des clés avec leurs valeurs correspondantes, à l'exception des clés numériques qui seront innaccessibles à moins d'être itérées.

```
$obj = (object) array('1' => 'foo');
var_dump(isset($obj->{'1'})); // affiche 'bool(false)'
var_dump(key($obj)); // affiche 'int(1)'
```

Pour n'importe quel autre type, un membre appelé scalar contiendra la valeur.

```
$obj = (object) 'ciao';
echo $obj->scalar; // Affiche : 'ciao'
```

Autre technique peu connue, et qui peut rendre de grands services : l'objet stdClass permet de créer des objets dynamiquement :

```
$page = new stdClass();
$page->name='Home';
$page->status=1;

echo $page->name ;
```

Voici un article intéressant, concernant stdClass et la possibilité de « caster » un tableau en objet :

<http://www.webmaster-source.com/2009/08/20/php-stdclass-storing-data-object-instead-array/>

Suppose you wanted to quickly create a new object to hold some data about a book. You would do something like this:

```
$book = new stdClass();
$book->title = "Harry Potter and the Prisoner of Azkaban";
$book->author = "J. K. Rowling";
$book->publisher = "Arthur A. Levine Books";
$book->amazon_link = "http://www.amazon.com/dp/0439136369/" ;
```

You can then access the data by calling `$book->title` and so on.

Or what if you wanted to take an array and turn it into an object ? You can do that by casting the variable.

```
$array = array(
    "title" => "Harry Potter and the Prisoner of Azkaban",
    "author" => "J. K. Rowling",
    "publisher" => "Arthur A. Levine Books",
    "amazon_link" => "http://www.amazon.com/dp/0439136369/"
);

$books = (object) $array;
```

This should produce the same result as the previous code snippet. You could also go the other direction, casting an object into an array by using `$array = (array) $object`.

Objects are really useful when you're working with a large data structure, as you can have an object with nested sub-arrays in it, as SimpleXML tends to return. You get most of the data as properties of the result set object, and the posts in an RSS feed are assigned to array entries, which then have further objects inside them.

4.5.7 Espaces de nommage

PHP 5.3 introduit la possibilité de déclarer des espaces de noms (en anglais « namespaces ») pour catégoriser les classes PHP, les constantes et les fonctions. L'utilisation des espaces de noms permet d'éviter les collisions et de fournir un contexte à chaque code PHP.

En tant que développeur de l'application, vous pouvez utiliser des composants écrits par d'autres personnes. Vous ne pouvez jamais être sûr que ces personnes n'utilisent pas les mêmes noms de classes que vous, et cela pour un usage très différent. Avant l'arrivée des espaces de noms, le développeur qui voulait se protéger contre le risque de collisions était tenté de définir le contexte dans le nom de la classe, comme par exemple `My_Enterprise_Person` ou `XML_Validator`.

Exemple de déclaration d'une classe `Foo` à l'intérieur du namespace `MyApp` :

```
namespace MyApp;  
class Foo {  
    ...  
}
```

Exemple d'utilisation de la classe `Foo` liée au namespace `MyApp`

```
$foo = new \MyApp\Foo();
```

Définir une politique d'espace de noms avant de démarrer un projet est une bonne idée, même s'il est toujours possible - mais aussi plus délicat - de le faire dans un second temps.

Si vous construisez des applications pour une organisation, vous pouvez par exemple utiliser le nom de cette organisation au plus haut niveau du namespace. C'est généralement suffisant pour éviter les conflits de noms entre des classes du projet et des classes issues d'autres projets.

Subnamespaces

Les Subnamespaces sont des sous-espaces de noms à l'intérieur d'espaces de noms de niveau supérieur. Ils permettent de clarifier le contexte, si l'espace de nommage de niveau supérieur n'est pas suffisant pour établir le contexte de la classe PHP que vous créez ou utilisez.

Il convient d'être prudent dans l'utilisation des subnamespaces, et de se limiter à une dizaine de subnamespaces (voire moins), car au delà l'organisation du code peut devenir particulièrement complexe.

Exemple de déclaration de subnamespace à l'intérieur d'un espace de noms de haut niveau :

```
namespace ACME\ServicePersonnel;
```

Bonnes pratiques de nommage.

On peut découper ses espaces de noms par projet, ou encore par domaine, avec éventuellement un découpage en sous-domaines, chaque sous-domaine définissant un projet. Il est fréquent de disposer de classes qui sont communes à différents projets ou domaines. On peut dans ce cas créer un sous-domaine "Common" comme dans l'exemple ci-dessous :

Exemple d'utilisation d'un subnamespace "common" au sein d'un "namespace" de plus haut niveau

```
namespace MyCompany\Common\Validation;  
  
class NotNullValidator {  
  
    // ...  
}
```

Utilisation d'alias

Bien que les espaces de noms facilitent l'organisation du code, ils ont souvent pour inconvénient d'entraîner l'écriture d'espaces de noms (incluant des subnamespaces) à rallonge. Pour éviter cela, on peut recourir à des alias, comme dans l'exemple ci-dessous.

```
use MyCompany\Common\Validation as Validators;  
  
$test = new \Validators\MyClass ;
```

Dans l'exemple ci-dessus, on a profité de la définition de l'alias pour lui attribuer un nom différent (Validators au lieu de Validation), mais on n'est pas obligé de procéder ainsi.

Exemple de namespace sans alias :

```
$response = new \Symfony\Component\HttpFoundation\Response('Oops', 400); $response->send();
```

... le même exemple cette fois-ci avec alias (sans renommage) :

```
use Symfony\Component\HttpFoundation\Response;  
  
$response = new Response('Oops', 400); $response->send();
```

A partir de PHP 5.6, il est possible d'importer des fonctions et des constantes.

Pour importer une fonction, remplacer « use » par « use func » :

```
use func Namespace\functionName;  
  
functionName();
```

Pour importer une constante, remplacer « use » par « use constant » :

```
use constant Namespace\CONST_NAME;  
  
echo CONST_NAME;
```

Il est possible d'importer plusieurs namespaces dans l'espace de nommage courant en procédant de la façon suivante :

```
use Symfony\Component\HttpFoundation\Request,  
      Symfony\Component\HttpFoundation\Response,  
      Symfony\Component\HttpFoundation\Cookie;
```

Mais le code obtenu peut vite devenir confus, aussi on recommandera de déclarer chaque namespace sur un « use » distinct :

```
use Symfony\Component\HttpFoundation\Request;  
  
use Symfony\Component\HttpFoundation\Response;
```

```
use Symfony\Component\HttpFoundation\Cookie;
```

Pour approfondir le sujet :

<http://php.net/manual/fr/language.namespaces.php>

4.5.8 SPL (Standard PHP Library)

PHP est livré avec une bibliothèque de classes standard, la SPL.

<http://php.net/manual/fr/book.spl.php>

Je vous laisse le soin de parcourir la liste des classes proposées par la SPL.

Je vous propose ci-dessous un exemple de script PHP faisant appel à l'une des classes de la SPL, la classe DirectoryIterator. En s'appuyant sur cette classe, le script parcourt le contenu d'un répertoire, et renvoie un tableau des fichiers que ce répertoire contient. Ce petit script m'a rendu service à plusieurs reprises :

```
/** 
 * Fonction qui permet de récupérer l'extension d'un fichier
 * nécessaire pour les versions de PHP < 5.3.6
 * @param type $fic
 * @return type
 */
function recuperExtension($fic) {
    /*
     * on sépare toutes les parties du nom du fichier en utilisant
     * le point comme séparateur et on stocke ces parties dans un tableau
     */
    $ext = explode(".", $fic);
    // l'extension est contenue dans le dernier poste du tableau
    $nb = count($ext);
    if ($nb > 0) {
        return strtolower($ext[$nb - 1]);
    } else {
        return '';
    }
}

function mktime2datetime($mkmtime) {
    return date('Y-m-d H:i:s', $mkmtime);
}

$path = 'D:\Tools';

$dir = new DirectoryIterator($path);
$files = array();
foreach ($dir as $entry) {
    $filename = $entry->getFilename();
```

```
if ((!$entry->isDot()) && ($filename != '~')  
    && ($filename != basename($_SERVER['PHP_SELF']))) {  
    if ($entry->getType() == 'file') {  
        if (version_compare(PHP_VERSION, '5.3.6') >= 0) {  
            $extension = $entry->getExtension();  
        } else {  
            $extension = recuperExtension($filename);  
        }  
    } else {  
        $extension = '';  
    }  
  
    $files [] = array (  
        "name" => $filename,  
        "size" => $entry->getSize (),  
        "type" => $entry->getType (),  
        "mtime" => $entry->getMtime (),  
        "datetime" => mktimes2datetime($entry->getMtime ()),  
        "path" => $entry->getPathname (),  
        "read" => $entry->isReadable (),  
        "write" => $entry->isWritable (),  
        "exec" => $entry->getType () == 'file' ? $entry->isExecutable () : '',  
        "ext" => $extension );  
    }  
}  
  
// supprimer la ressource  
unset($dir);  
echo '<pre>';  
var_dump($files);
```

4.5.9 Design Patterns

Les premiers modèles de conception (en anglais "design patterns") ont été élaborés, au milieu des années 90, par Erich Gamma, Richard Helm, Ralph Johnson , et John Vlissides, familièrement désignés sous le nom de « bande des quatre », en anglais, "Gand of Four" (GOF).

http://fr.wikipedia.org/wiki/Design_Patterns

La bande des quatre publia en 1994 le livre suivant :

Design Patterns: Elements of Reusable Object-Oriented Software (pub. Addison-Wesley)

Traduit en français peu de temps après (plutôt mal traduit d'ailleurs) sous le titre « *Design patterns. Catalogue des modèles de conception réutilisables* », ce livre est un livre de méthodologie appliquée à la conception logicielle.

L'idée derrière les « design patterns » est simple. Au cours de leurs années de développement de logiciels, les membres de la "bande des quatre" se sont aperçus qu'un certain nombre de modèles de conception émergeaient de leur expérience du développement logiciel. Disposer de ces modèles de conception, signifiait qu'ils pouvaient concevoir des applications plus robustes plus rapidement.

L'élaboration de ces modèles de conception, et le fait de les désigner par des noms distincts a permis de faciliter le dialogue et les échanges de bonne pratiques entre développeurs.

Ces modèles de conception peuvent être utilisés aussi bien sur de gros que sur de petits projets. Mais il faut souligner que ces design patterns ont été conçus pour des langages compilés comme Java, et qu'ils ne sont pas tous adaptés aux langages interprétés comme PHP.

Pour une présentation détaillée de 11 design patterns (adaptés à PHP), je recommande vivement la lecture des 2 articles suivants écrits par Jack Herrington :

<https://www.ibm.com/developerworks/library/os-php-designptrns/>

<https://www.ibm.com/developerworksopensource/library/os-php-designpatterns/>

Il faut souligner que le développeur d'applications web, n'a pas nécessairement besoin de maîtriser l'ensemble des design patterns. Mais il sera très certainement amené à utiliser des frameworks PHP qui eux-mêmes implémentent des design patterns, et une connaissance minimale du sujet pourra lui être utile.

On trouvera un exemple de design pattern dans ce cours, au chapitre 4.6.2, avec une classe de type Singleton (la classe DBConnect).

4.5.10 Quoi d'autre ?

Le langage PHP s'est enrichi au fil des années de nombreuses fonctionnalités, qui ne sont pas présentées dans ce cours d'introduction (vous pourrez les approfondir par la suite). Parmi ces fonctionnalités, on trouve :

- Late Static Binding (résolution statique à la volée) :
<http://php.net/manual/fr/language.oop5.late-static-bindings.php>
- Classes anonymes (apparues avec PHP7) :
<http://php.net/manual/fr/language.oop5.anonymous.php>
- La classe Closure : <http://php.net/manual/fr/class.closure.php>
- Les fonctions de rappel : <http://php.net/manual/fr/language.types.callable.php>
- Les traits : <http://php.net/manual/fr/language.oop5.traits.php>
- Les générateurs : <http://php.net/manual/fr/language.generators.overview.php>

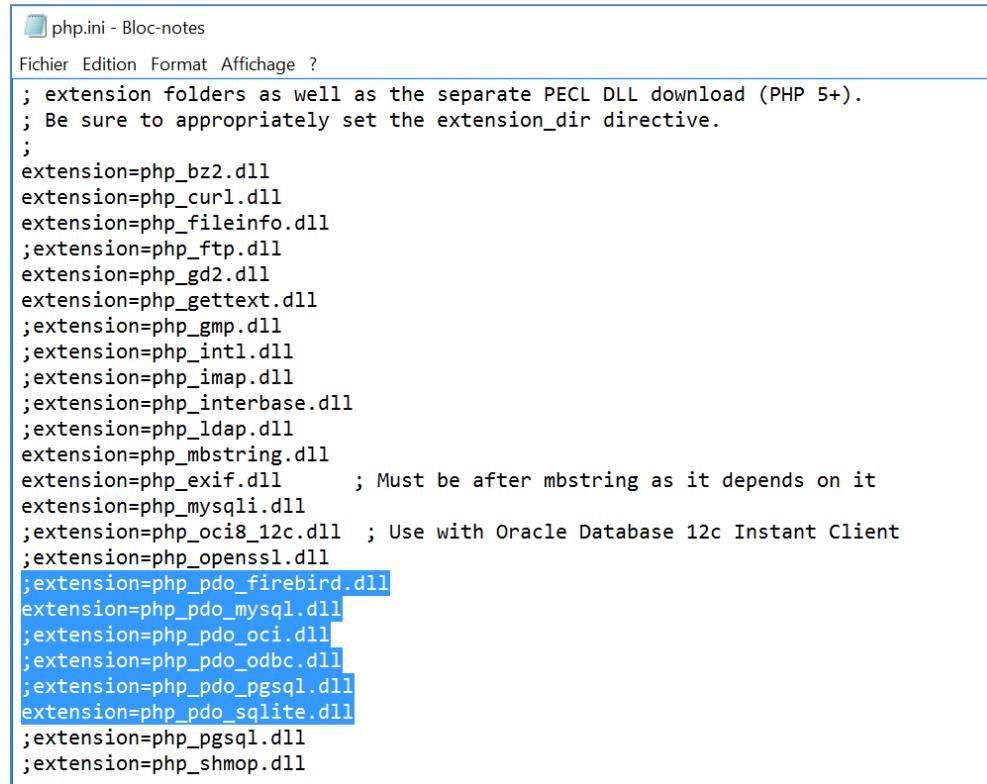
4.6 PHP et le SQL

4.6.1 Présentation de PDO

PDO = PHP Data Object

PDO est une extension PHP qui fournit une interface permettant de se connecter à différentes bases de données au travers d'un même jeu de fonctions PHP, là où auparavant chaque base de données nécessitait un driver et un jeu de fonctions PHP spécifiques.

Pour que PDO fonctionne avec une base de données particulière, il faut que l'extension correspondante soit activée dans le fichier de configuration php.ini de votre stack PHP :



```
php.ini - Bloc-notes
Fichier Édition Format Affichage ?
; extension folders as well as the separate PECL DLL download (PHP 5+).
; Be sure to appropriately set the extension_dir directive.
;
extension=php_bz2.dll
extension=php_curl.dll
extension=php_fileinfo.dll
;extension=php_ftp.dll
extension=php_gd2.dll
extension=php_gettext.dll
;extension=php_gmp.dll
;extension=php_intl.dll
;extension=php_imap.dll
;extension=php_interbase.dll
;extension=php_ldap.dll
extension=php_mbstring.dll
extension=php_exif.dll ; Must be after mbstring as it depends on it
extension=php_mysqli.dll
;extension=php_oci8_12c.dll ; Use with Oracle Database 12c Instant Client
;extension=php_openssl.dll
;extension=php_pdo_firebird.dll
extension=php_pdo_mysql.dll
;extension=php_pdo_oci.dll
;extension=php_pdo_odbc.dll
;extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
;extension=php_pgsql.dll
;extension=php_shmop.dll
```

Dans l'exemple ci-dessus, les extensions actives sont celles qui n'ont pas de point virgule en première position, comme par exemple l'extension « `php_pdo_mysql` », qui est active par défaut.

Pour accéder à une base MySQL, on pourrait utiliser l'extension « `php_mysqli` » plutôt que PDO, mais PDO est beaucoup plus pratique, aussi je ne parlerai pas ici de l'extension « `php_mysqli` ».

4.6.2 Connexion PDO et premiers exemples

L'ouverture d'une connexion à une base de données avec PDO se fait en instanciant un objet avec la classe PDO.

Deux syntaxes déclaratives permettant de faire la même chose :

- syntaxe n°1 :

```
<?php
// MySQL expects parameters in the string
$user = "root";
$password = "";
$db = new PDO('mysql:host=localhost;dbname=nomdelabase', $user, $password);

$st = $db->query('SELECT id, nom_categ_fr FROM produit_categorie');
foreach ($st->fetchAll() as $row) {
    print "id = {$row['id']} , nom = {$row['nom_categ_fr']} <br/>\n";
}
```

- syntaxe n°2 :

```
<?php
//paramètres de connexion
$host = 'localhost'; // le chemin vers le serveur
$db = 'nomdelabase'; // le nom de la base de donnée
$user = 'root'; // nom d'utilisateur pour se connecter
$pwd = ''; // mot de passe de l'utilisateur pour se connecter
$dsn = "mysql:dbname=$db;host=$host"; //creation du dsn pour mysql

//utilisation de la base
try {
    //Ouverture de l'accès à la base
    $pdo= new PDO($dsn, $user, $pwd);
    //echo "connectee a la base";
} catch(PDOException $e){
    echo 'Erreur : '.$e->getMessage().'  
';
    echo 'N° : '.$e->getCode();
    die('Echec de la connexion : '.$e->getMessage());
} ;

$st = $pdo->query('SELECT id, nom_categ_fr FROM produit_categorie');
foreach ($st->fetchAll() as $row) {
    print "id = {$row['id']} , nom = {$row['nom_categ_fr']} <br/>\n";
}
```

=> Résultat obtenu dans le navigateur (identique dans les 2 syntaxes) :

```
id = 1 , nom = Distributeurs Anti-limaces
id = 2 , nom = Spécial déchaumeur
id = 3 , nom = Saleuses électriques
id = 4 , nom = Microgranulateur électrique
id = 5 , nom = Semoir à engrais
id = 6 , nom = Agrainoir mobile
id = 0 , nom = _Sans Catégorie
id = 13 , nom = Doseur électrique
```

Quelques techniques propres à MySQL à noter :

- pour récupérer l'identifiant d'une ligne que l'on vient d'insérer, on utilise la technique suivante :

```
$sql = "SELECT LAST_INSERT_ID() AS lastid ";
$stmt2 = $conn->query($sql);
$row = $stmt2->fetch(PDO::FETCH_LAZY);
echo 'ID of last insert: ', $row->lastid;
```

- pour travailler en UTF-8 avec MySQL, il est impératif d'exécuter en début de session la requête suivante :

```
SET NAMES 'utf8';
```

Si vous travaillez avec PostgreSQL, la syntaxe sera légèrement différente, mais ce sera le même principe :

```
SET CLIENT_ENCODING TO 'UTF8';
```

Quand j'indiquais plus haut qu'il fallait exécuter cette instruction en « début de session », je disais cela du double point de vue PHP et SQL. De ce point de vue, une « session » regroupe l'ensemble des instructions SQL qui vont être exécutées dans un script, entre l'ouverture de la connexion à la base de données, et la fermeture de cette même connexion. Donc l'activation de l'UTF-8 se fera juste après l'ouverture de la connexion, et sera ainsi valable pour toutes les requêtes exécutées durant la session :

```
$conn = new PDO($dsn, $username, $password);
// Passage en mode UTF-8 systématique
$conn->exec("SET NAMES 'utf8'");
```

Voici un exemple de classe générique pouvant être utilisée pour simplifier l'ouverture des connexions bases de données. On y retrouve le forçage du mode FETCH_ASSOC par défaut, et l'activation de l'UTF-8 (avec la syntaxe MySQL), également par défaut :

```
class DBConnect {
    private static $dbName = 'crud_tutorial';
    private static $dbHost = 'localhost';
    private static $dbUsername = 'root';
    private static $dbUserPassword = '';

    private static $cont = null;

    public function __construct() {
        die('Init function is not allowed');
    }

    public static function open() {
        // One connection through whole application
        if ( null == self::$cont )
        {
            try
            {
                $dsn = "mysql:host=".self::$dbHost.";"."dbname=".self::$dbName;
                self::$cont = new PDO($dsn, self::$dbUsername, self::$dbUserPassword);
                // Force un mode de gestion plus fin des erreurs SQL
                self::$cont->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
                // Force le mode FETCH_ASSOC comme mode par défaut
                self::$cont->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE,
                    PDO::FETCH_ASSOC);
                // Passage en mode UTF-8 systématique
                self::$cont->exec("SET NAMES 'utf8'");
            }
            catch(PDOException $e)
            {
                die($e->getMessage());
            }
        }
        return self::$cont;
    }

    public static function close() {
        self::$cont = null;
    }
}
```

En plaçant cette classe dans un script à part, voici un exemple d'utilisation :

```
require_once 'library/DBConnect.php';

// création d'une instance de PDO
$db = DBConnect::open();
```

On trouvera différents exemples d'utilisation de cette classe dans le chapitre 4.9 (études de cas).

4.6.3 Les différents modes Fetch de PDO

La méthode **fetchAll()** permet de renvoyer un jeu de données complet en une seule passe (cf. exemple en début de chapitre).

La méthode **fetch()** permet d'adresser chaque enregistrement individuellement :

Les méthodes **fetch()** et **fetchAll()** acceptent des paramètres complémentaires qui sont :

Constante	Format de ligne
PDO::FETCH_NUM	Tableau avec clés de type numérique (en fait les numéros de colonnes du Result Set)
PDO::FETCH_ASSOC	Tableau avec clés de type chaîne (en fait les noms de colonnes du Result Set)
PDO::FETCH_BOTH	Format par défaut si rien n'est précisé. Mixe les constantes PDO::FETCH_NUM et PDO::FETCH_ASSOC
PDO::FETCH_OBJ	Objet de classe stdClass avec noms de colonnes repris en noms de propriétés.
PDO::FETCH_LAZY	Object de classe PDORow avec noms de colonnes repris en noms de propriétés. Les propriétés ne sont générées que si elles font du Result Set, toute propriété qui est addressée alors qu'elle n'est pas encore créée déclenche une erreur. C'est donc un bon choix si la ligne du Result Set contient beaucoup de colonnes.
PDO::FETCH_BOUND	Le mode « fetch » permet de définir des variables dont les valeurs sont regénérées à chaque nouvel appel de <i>fetch()</i>
PDO::FETCH_INTO	Permet de placer les lignes du Result Set dans des objets spécialisés de classes particulières.
PDO::FETCH_CLASS	Pour employer ces modes, il faut au préalable créer une classe qui prolonge la classe intégrée de PDOStatement

Je rappelle qu'il est possible de forcer un mode fetch par défaut lors de l'ouverture de la session PDO, cela évite d'avoir à préciser ce mode sur chaque requête SQL. C'est ce que j'ai fait dans la classe DBConnect présentée au chapitre précédent (cf. extrait de code ci-dessous) :

```
// Force le mode FETCH_ASSOC comme mode par défaut
self::$cont->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
```

Pour ma part, le mode que je préfère c'est le mode **FETCH_ASSOC**. C'est celui qui me permet d'obtenir des tableaux associatifs faciles à manipuler, comme dans cet exemple emprunté à une de nos études de cas du chapitre 4.9 :

```

foreach($datas as $data) {
    // Formatage du prix selon la notation française
    $data['prix_vente'] = number_format($data['prix_vente'], 2, ',', ' ');
    echo <<<TABLEROW
<tr>
    <td>{$data['code_produit']}</td>
    <td>{$data['nom_famille']}</td>
    <td>{$data['nom_activite']}</td>
    <td class="text-center">{$data['date_tarif']}</td>
    <td class="text-right">{$data['prix_vente']}</td>
</tr>
TABLEROW;
}

```

Utiliser `PDO::exec()` pour envoyer un `INSERT`, un `DELETE`, ou un `UPDATE`, comme dans les exemples suivants :

```

$db->exec ("INSERT INTO friends (id, name) VALUES (1, 'Freddy')");

$db->exec ("DELETE FROM friends WHERE name = 'Sweeny'");

$db->exec ("UPDATE friends SET is_cool = 1 WHERE name = 'Jack'");

```

Utiliser `PDO::prepare()` et `PDOStatement::execute()` pour préparer et exécuter une requête en 2 temps :

```

$st = $db->prepare('INSERT INTO friends (id, name) VALUES (?, ?)');
$st->execute(array(1, 'Freddy'));

$st = $db->prepare('DELETE FROM friends WHERE name = ?');
$st->execute(array('Sweeny'));

$st = $db->prepare('UPDATE friends SET is_cool = ? WHERE name = ?');
$st->execute(array(1, 'Jack'));

```

La méthode `exec()` renvoie de la base de données ce que cette dernière lui transmet. Pour `INSERT`, `UPDATE` et `DELETE`, elle renvoie le nombre de lignes affectées par la requête.

Les méthodes `prepare()` et `execute()` sont utiles pour les requêtes qui doivent être exécutées plusieurs fois. On peut préparer une requête une fois, et l'exécuter plusieurs fois avec des paramètres différents (c'est très bon pour les performances ☺) :

```

$st = $db->prepare('DELETE FROM friends WHERE name LIKE ?');
$st->execute(array('Freddy'));

```

```
$st->execute(array('Sweeny'));
$st->execute(array('Jack'));
```

Autre exemple d'utilisation :

```
// Prepare
$st = $db->prepare("SELECT sign FROM zodiac WHERE element = ?");
// Execute once
$st->execute(array('fire'));
while ($row = $st->fetch()) {
    print $row[0] . "<br/>\n";
}
// Execute again
$st->execute(array('water'));
while ($row = $st->fetch()) {
    print $row[0] . "<br/>\n";
}
```

Exemple d'utilisation avec **2 paramètres marqueurs** :

```
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element = ? OR planet = ?");

// SELECT sign FROM zodiac WHERE element = 'earth' OR planet = 'Mars'
$st->execute(array('earth', 'Mars'));
$row = $st->fetch();
```

Exemple d'utilisation avec **2 paramètres nommés** :

```
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element = :element OR planet = :planet");
// SELECT sign FROM zodiac WHERE element = 'earth' OR planet = 'Mars'
$st->execute(array('planet' => 'Mars', 'element' => 'earth'));
$row = $st->fetch();
```

Exemple d'utilisation avec **la méthode BindParam** :

```
$pairs = array('Mars' => 'water',
              'Moon' => 'water',
              'Sun' => 'fire');
$st = $db->prepare(
    "SELECT sign FROM zodiac WHERE element = :element AND planet = :planet");
$st->bindParam(':element', $element);
$st->bindParam(':planet', $planet);
```

```

foreach ($pairs as $planet => $element) {
    // Aucun paramètre à transmettre dans ce cas à execute()
    $st->execute();
    var_dump($st->fetch());
}

```

Comptage de lignes avec la fonction RowCount() :

- pour les INSERT, UPDATE, et DELETE exécutés avec la fonction exec(), la valeur renvoyée par cette fonction contient le nombre de lignes modifiées (cf. exemple de la page précédente).
- pour les INSERT, UPDATE, et DELETE exécutés avec PDO::prepare() et PDOStatement::execute(), l'appel de la fonction PDOStatement::rowCount() permet de récupérer le nombre de lignes modifiées :

```

$st = $db->prepare('DELETE FROM family WHERE name LIKE ?');
$st->execute(array('Fredo'));
print "Deleted rows: " . $st->rowCount();
$st->execute(array('Sonny'));
print "Deleted rows: " . $st->rowCount();
$st->execute(array('Luca Brasi'));
print "Deleted rows: " . $st->rowCount();

```

- comptage du nombre de lignes renvoyées par un SELECT (on peut préférer un SELECT COUNT(*) à la méthode ci-dessous) :

```

$st = $db->query('SELECT symbol, planet FROM zodiac');
$all= $st->fetchAll(PDO::FETCH_COLUMN, 1);
print "Retrieved ". count($all) . " rows";

```

Quelques articles intéressants pour compléter ce chapitre :

<https://www.cloudconnected.fr/2007/10/10/les-fetch-modes-de-pdo/>

<https://www.cloudconnected.fr/2007/10/12/les-fetch-modes-de-pdo-2-les-modes-orientes-objet/>

<http://php.net/manual/fr/book pdo.php>

4.6.4 SQL et sécurité

La technique dite des « requêtes SQL paramétrées » constitue le socle que tout développeur SQL doit connaître pour produire un développement de qualité.

Les requêtes SQL paramétrées offrent tous les avantages :

- Sécurité : car elles protègent contre les attaques dites « par injection SQL »
- Souplesse d'écriture : nul besoin avec cette technique de concaténer laborieusement les paramètres à l'intérieur des requêtes SQL
- Performances : une requête paramétrée, si elle est exécutée plusieurs fois avec des valeurs différentes, va bénéficier du plan d'accès déterminé par le moteur SQL lors de la première exécution de la requête. Les exécutions suivantes d'une même requête seront dès lors beaucoup plus rapides (le chemin d'accès n'étant calculé qu'une fois).

La technique de l'attaque par injection SQL a été décrite dans de nombreux articles disponibles sur internet. Nous ne l'approfondirons pas ici, faute de temps, aussi je vous recommande la lecture des articles suivants :

- Wikipédia : http://fr.wikipedia.org/wiki/Injection_SQL
- PHP Security Consortium :
 1. <http://phpsec.org/library/>
 2. <http://unixwiz.net/techtips/sql-injection.html>

Voici un exemple de script utilisant une requête SQL paramétrée :

```
// requête SQL paramétrée avec deux "jokers" matérialisés par les ?
$sql = 'SELECT id, nom_categ_fr FROM produit_catégorie WHERE id >= ? AND id <= ?';
$params = array(1, 100);

$stmt = $db->prepare($sql); // préparation d'un "statement" à partir de la requête
$stmt->execute($params); // exécution de la requête avec ses 2 paramètres
if ($stmt) {
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    while ($row != false) {
        print "id = {$row['id']} , nom = {$row['nom_categ_fr']} <br>".PHP_EOL;
        $row = $stmt->fetch(PDO::FETCH_ASSOC);
    }
}
unset($stmt);
```

Nous étudierons d'autres exemples de requête SQL paramétrée dans le chapitre 4.9 (études de cas).

4.6.5 Connexion à différents SGBD

PDO permet d'accéder à différentes bases de données :

```
// MySQL reçoit ses paramètres de connexion dans une DSN (Data Source Name)
// les paramètres sont séparés par des points virgules (;)
$mysql = new PDO('mysql:host=db.example.com;port=31075 dbname=food', $user,
$password);

// Connexion à un serveur MySQL local sur Linux
$mysql = new PDO('mysql:unix_socket=/tmp/mysql.sock', $user, $password)

// PostgreSQL reçoit aussi ses paramètres de connexion dans une chaîne mais
// séparés par un blanc
$pgsql = new PDO('pgsql:host=db.example.com port=31075 dbname=food', $user,
$password);

// On peut placer "user" et "password" dans la DSN
$pgsql = new PDO("pgsql:host=db.example.com port=31075 dbname=food user=$user
password=$password");

// Avec Oracle
// si un nom de database est défini dans tnsnames.ora, on l'indique dans DSN
$oci = new PDO('oci:food', $user, $password);
// dans le cas contraire, on définit un "Instant Client URI"
$oci = new PDO('oci:dbname://db.example.com:1521/food', $user, $password);

// Sybase (si PDO utilise FreeTDS)
$sybase = new PDO('sybase:host=db.example.com;dbname=food', $user, $password);
// Microsoft SQL Server (si PDO utilise les librairies MS SQL Server)
$mssql = new PDO('mssql:host=db.example.com;dbname=food', $user, $password);

// ODBC avec une connexion prédéfinie
$odbc = new PDO('odbc:DSN=food');
// ODBC avec une connexion définie dans la DSN (nécessite un driver)
$odbc = new PDO('odbc:Driver={Microsoft Access Driver
(*.mdb)};DBQ=C:\\data\\food.mdb;Uid=Chef');

// SQLite attend un simple nom de fichier (pas de "user" ni de "password")
$sqlite = new PDO('sqlite:/usr/local/zodiac.db');
$sqlite = new PDO('sqlite:c:/data/zodiac.db');

// SQLite peut aussi manipuler en mémoire des bases temporaires
$sqlite = new PDO('sqlite::memory:');
// SQLite DSN (v2 et v3)
$sqlite2 = new PDO('sqlite2:/usr/local/old-zodiac.db');
```

Présentation exhaustive des paramètres de connexion ODBC, classés par bases de données :
<https://www.connectionstrings.com/>

Problème de socket lors de la connexion à MySQL avec PDO sous Linux :

En environnement LAMP, il arrive – y compris avec Zend Server – de rencontrer l'erreur suivante lors de la connexion :

Fatal error: Uncaught exception 'PDOException' with message 'SQLSTATE[HY000] [2002]

[SQL Errcode 2002] SQLSTATE[HY000] [2002] Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)

Cela signifie que le connecteur a besoin qu'on lui précise au niveau du DSN, soit l'adresse TCP/IP (solution 1), soit l'emplacement du socket Linux pour MySQL (solution 2) :

1: "mysql:host=127.0.0.1" or "mysql:host=localhost;port=3306"

2: "mysql:unix_socket=/var/run/mysqld/mysqld.sock"

A noter que la solution n°2 fonctionne bien sous Zend Server.

Une troisième solution a été évoquée dans les forums, qui n'a pas semblé fonctionner avec Zend Server, mais je l'ai quand même relevée ci-dessous, pour mémoire : elle consiste à modifier le fichier php.ini en spécifiant la ligne suivante (exemple fourni pour XAMPP) :

pdo_mysql.default_socket = /opt/lampp/var/mysql/mysql.sock

En environnement Zend Server, la ligne à insérer serait plutôt :

pdo_mysql.default_socket = /var/run/mysqld/mysqld.sock

Il ne faut pas oublier de redémarrer Apache après toute modification de php.ini.

4.6.6 What else ?

Existe-t-il d'autres manières d'accéder à une base de données qu'en écrivant du SQL et le passant au SGBD via PDO ou des fonctions dédiées ?

Oui bien sûr, il existe en particulier des outils de type ORM (Object Relational Mapping), tels que les projets Doctrine, Propel et quelques autres.

Voici quelques liens à explorer :

[https://fr.wikipedia.org/wiki/Doctrine_\(ORM\)](https://fr.wikipedia.org/wiki/Doctrine_(ORM))

<https://fr.wikipedia.org/wiki/Propel>

<https://laravel.com/docs/5.4/eloquent> (ORM spécifique à Laravel)

<https://github.com/atlasphp/AtlasOrm> (par Paul M. Jones, auteur de plusieurs livres intéressants disponibles chez Leanpub.com).

<http://redbeanphp.com/index.php>

<https://github.com/bcosca/fatfree>

On trouve une bonne introduction à Doctrine sur OpenClassRooms :

<https://openclassrooms.com/courses/utilisation-d-un-orm-les-bases-de-doctrine>

L'usage des ORM est très controversé, comme vous pourrez le constater en lisant ces quelques pages :

<https://php.developpez.com/actu/61207/-ORM-or-not-ORM-faut-il-les-utiliser-ou-continuer-a-ecrire-simplement-des-requetes-SQL/>

<https://stackoverflow.com/questions/2169832/data-mapper-vs-active-record>

On trouvera un exemple d'utilisation de Doctrine dans le cours relatif au framework SILEX :

https://github.com/gregja/Cours_SILEX_4_Biz

4.7 Composer

TODO : en cours de réécriture

4.8 Les fichiers

4.8.1 Les fichiers texte

Ouverture/Lecture/Mise à jour/Fermeture de fichiers

Exemples d'ouverture de fichier :

```
// ouvrir "filename" en lecture
$filehandle = fopen("filename", "r");
// ouvrir "filename" en écriture
$filehandle = fopen("filename", "w");
// ouvrir "filename" en écriture binaire
$filehandle = fopen("filename", "wb");
```

Fermeture d'un fichier :

```
fclose($filehandle);
```

Lire un fichier :

```
// Lire une ligne du fichier lié à $filehandle
$string = fgets($filehandle);
// Lire un caractère du fichier lié à $filehandle
$char = fgetc($filehandle);
// Lit un morceau d'octets du fichier lié à $filehandle
$text = fread($filehandle, $bytes );
// Lit l'intégralité du fichier "filename.txt" dans une "string"
$text = file_get_contents("filename.txt");
// Lit l'intégralité du fichier "filename.txt" et le renvoie dans un tableau
$text = file("filename.txt");
```

Ecrire dans un fichier :

```
// Ecrit $string dans le fichier lié à $filehandle
fwrite($filehandle, $string);
// Ecrit $string dans le fichier "filename"
file_put_contents("filename", $string);
```

Attention, la plupart des fonctions présentées ici utilisent des paramètres optionnels, et je vous recommande de lire leur documentation sur php.net pour pouvoir les utiliser de manière optimale.

On étudiera un exemple d'utilisation de certaines de ces fonctions, au chapitre suivant traitant des fichiers CSV.

Tiens, avant de passer au chapitre sur les fichiers CSV, voici un exemple de script que j'ai retrouvé dans mes archives :

```
<?php
// Lit un fichier d'adresses
$lines = file('adresses.txt');
// Affichage des adresses
foreach ($lines as $line_num => $line) {
    $nom = substr($line, 10, 35);
    $rue1 = substr($line, 45, 35);
    $rue2 = substr($line, 80, 35);
    $pays = substr($line, 115, 2);
    $dept = substr($line, 117, 2);
    $cpos = substr($line, 132, 9);
    $vill = substr($line, 141, 35);

    echo "Ligne #<b>{$line_num}</b> : " ;
    echo htmlspecialchars($nom) . ', ' ;
    echo htmlspecialchars($rue1) . ', ' ;
    echo htmlspecialchars($rue2) . ', ' ;
    echo htmlspecialchars($cpos) . ', ' ;
    echo htmlspecialchars($vill) . ', ' ;
    echo htmlspecialchars($pays) . ', ' ;
    echo htmlspecialchars($dept) ;
    echo "<br />".PHP_EOL;
}
}
```

L'objectif de ce script, c'est de lire un fichier texte « plat », c'est-à-dire un fichier dans lequel les données de chaque ligne sont placées à des positions fixes (et qui sont les mêmes pour toutes les lignes). Voici un échantillon du fichier en question :

1	VNNRIV	VOHNSON SCREENS	ZONE INDUSTRIELLE	ZONE INDUSTRIELLE	FR86	0201002010026530	AVAILLES EN CHATEUDEAU
2	VNNRIV	LES PORTES DU FUTUR	28 ALLEE DU HAUT SERSOU	MR BOURDIN LIONEL	FR86	0345678906286360	CHASSENEUIL DU SERSOU
3	VNNRIV	VELO CHATELLEDIAULT	CC LECLERC	144, AVENUE FOCH	FR86	0303030303030300	CHATEAUDEAULT
4	VNNRIV	SPAGNETTI RAVIOLI	ZI VALLEE D'ARGENTON		FR86	0040404040404100	CHATEAUDEAUL
5	VNNRIV	MAGNETTI MARELLI	ZI VALLEE D'ARGENTON		FR86	0305050505056100	CHATEAUDEAUL

Bon, quand vous tombez sur un fichier de ce genre, et que vous devez en extraire des informations, il faut s'armer de patience, et surtout d'un bon éditeur de code qui vous donne la position précise de chaque caractère. A l'arrivée,

Ligne #0 : VOHNSON SCREENS , ZONE INDUSTRIELLE , ZONE INDUSTRIELLE , 26530 , AVAILLES EN CHATEUDEAU , FR, 86
 Ligne #1 : LES PORTES DU FUTUR , 28 ALLEE DU HAUT SERSOU , MR BOURDIN LIONEL , 86360 , CHASSENEUIL DU SERSOU , FR, 86
 Ligne #2 : VELO CHATELLEDIAULT , CC LECLERC , 144, AVENUE FOCH , 30300 , CHATEAUDEAULT , FR, 86

Si vous êtes vous-même obligé de créer un fichier plat pour répondre aux besoins d'un utilisateur ou d'un client, alors je vous recommande de regarder du côté des fonctions printf() et sprintf(), qui offrent beaucoup de souplesse dans le formatage des données affichées. Voici un exemple emprunté à la documentation php.net de la fonction sprintf() :

```
$s = 'monkey';
$t = 'many monkeys';
printf("%s\n",      $s); // standard string output
//=> [monkey]
printf("%10s\n",    $s); // right-justification with spaces
//=> [ monkey]
printf("%-10s\n",   $s); // left-justification with spaces
//=> [monkey ]
printf("%010s\n",   $s); // zero-padding works on strings too
//=> [0000monkey]
printf("%#10s\n",   $s); // use the custom padding character '#'
// => [###monkey]
printf("%10.10s\n", $t); // left-justification but with a cutoff of 10 characters
//=> [many monke]
```

Pour de plus amples informations sur le sujet :

<http://php.net/manual/fr/function.sprintf.php>

4.8.2 Les fichiers CSV

Le format CSV est un vieux format de fichier, qui est encore souvent utilisé en entreprise pour des échanges simples de données entre utilisateurs et applications. Il est par exemple très facile dans les tableurs comme Excel (de Microsoft) ou son équivalent sur Libre Office, d'exporter et d'importer des données au format CSV.

Le format CSV est un format de fichier texte, il est facilement lisible par un humain. Mais ce n'est pas un format normalisé, aussi on trouve de petites différences d'un fichier à l'autre. Les principales différences sont facilement lisibles dans les 2 jeux de données ci-dessous.

Voici un extrait de fichier CSV :

```
"AE","EMIRATS ARABIE","EMIRATS ARABIE",0,"",19990819,"ICAR1"
"AF","AFGHANISTAN","AFGHANISTAN",0,"*",19980715,"TVB"
"AG","ANTIGUA AND BARBUDA","ANTIGUA AND BARBU",0,"",19980715,"TVB"
"AI","ANGUILLA","ANGUILLA",0,"",19980715,"TVB"
"AN","LES ANTILLES","LES ANTILLES",0,"",19990819,"ICAR1"
"CF","CENTRE AFRIQUE","CENTRE AFRIQUE",0,"",19990819,"ICAR1"
```

Voici un extrait provenant d'un autre fichier CSV :

```
AE;"EMIRATS ARABIE";"EMIRATS ARABIE";0;;19990819;ICAR1
AF;AFGHANISTAN;AFGHANISTAN;0;*;19980715;TVB
AG;"ANTIGUA AND BARBUDA";"ANTIGUA AND BARBU";0;;19980715;TVB
AI;ANGUILLA;ANGUILLA;0;;19980715;TVB
AN;"LES ANTILLES";"LES ANTILLES";0;;19990819;ICAR1
CF;"CENTRE AFRIQUE";"CENTRE AFRIQUE";0;;19990819;ICAR1
```

Dans les 2 cas ci-dessus, il s'agit du même jeu de données (en l'occurrence un extrait d'une liste de pays). Mais dans le premier cas, chaque champ (ou colonne) est séparé de ses voisins par des virgules. Et les données de type texte sont systématiquement encadrés d'apostrophes doubles. Dans le second cas, les données sont séparées par des points virgules, et seules les chaînes de caractères composées de plusieurs mots sont encadrées par des apostrophes doubles.

Que se passe-t-il si vous recevez un fichier CSV dans le premier format ci-dessus, mais que le logiciel dans lequel vous devez importer ce jeu de données reconnaît le second format. Eh bien, vous pouvez écrire un petit script PHP qui va exploiter les fonctions PHP fgetcsv() et fputcsv() pour lire et réencoder les données dans le bon format. C'est ce que fait le petit script PHP ci-dessous.

```

/*
 * copie d'un CSV dans un autre
*/
// forçage d'une durée limite d'exécution supérieure à la limite standard
set_time_limit ( 360 );

// récupération du nom de répertoire courant
$repert = dirname ( __FILE__ );

$fichier_input = $repert . '/csvpays.csv';
$fichier_output = $repert . '/csvpays2.csv';

$file_in = @fopen ( $fichier_input, "r" );
$file_out = @fopen ( $fichier_output, "w" );

if (is_resource ( $file_in ) && is_resource ( $file_out )) {
    while ( ! feof ( $file_in ) ) {
        $fields = fgetcsv ( $file_in, 0, ',', '"' );
        if (is_array ( $fields )) {
            fputcsv ( $file_out, $fields, ';', '"' );
        }
    }
}
if (is_resource ( $file_in )) {
    fclose ( $file_in );
}
if (is_resource ( $file_out )) {
    fclose ( $file_out );
}

```

Attention : il peut arriver que votre fichier CSV contienne des données encodées en UTF-8, mais que le logiciel dans lequel vous devez importer les données ne supporte pas cet encodage. Dans ce cas, vous pouvez insérer juste avant l'appel de la fonction fputcsv() une petite boucle parcourant le tableau \$fields, et appliquant à chaque poste du tableau la fonction utf8_decode().

4.8.3 Les fichiers XML

Le format XML est un format de données très utilisé en entreprise, pour les échanges de données entre applications. On le retrouve notamment dans les services webs utilisant le protocole SOAP.

Il était aussi beaucoup utilisé dans les premiers échanges client-serveur de type AJAX, mais comme le format XML n'est pas très facile à exploiter en Javascript, il est aujourd'hui largement supplanté par le format JSON (pour ce type d'usage spécifiquement).

Le format XML étant un format texte, il est lisible par les humains. S'il est plus verbeux que le CSV, il offre l'avantage de contenir à la fois les données mais aussi leur signification, grâce aux balises encapsulant les données, dont les noms sont généralement explicites.

Voici un exemple de fichier XML utilisé pour générer un menu dans une application web :

```
<?xml version="1.0" encoding="UTF-8"?>
<menus>
    <menu id="home">
        <id>home</id>
        <parent>home</parent>
        <title>Accueil</title>
        <url>index.php</url>
    </menu>
    <menu id="applications">
        <id>applications</id>
        <parent>home</parent>
        <title>Applications</title>
        <url>mnu_applications.php</url>
    </menu>
</menus>
```

L'indentation des données n'est pas obligatoire dans un fichier XML, mais elle facilite la lecture et la compréhension du jeu de donnée.

Nous avons ici une liste d'options de menus matérialisée par deux balises « menu » contenues dans une balise de niveau supérieur « menus ». Chaque option de menu a un identifiant matérialisé par la balise « id » et l'attribut « id » de la balise « menu ». Les deux identifiants sont identiques et donc redondants, la balise « id » pourrait par exemple être supprimée ce qui allègerait la taille du fichier. Il y a aussi une balise « parent », on voit que la première option de menu est liée à elle-même, et que la seconde option est liée à la première (la valeur de sa balise « parent » étant « home »).

Ce lien hiérarchique entre les 2 options de menus est spécifique à ce jeu de données, on trouvera souvent des jeux de données dans lesquels il n'y a pas de lien hiérarchique entre les différentes données du fichier XML.

4.8.3.1 Générer du XML

Exemple de création d'un menu en XML à partir d'un tableau PHP associatif multi-niveaux :

```
// force PHP à envoyer un flux de données XML plutôt que HTML
header('Content-Type: text/xml');
print '<?xml version="1.0"?>' . PHP_EOL;
print "<menus>". PHP_EOL;
$menus = array(
    "home" => array("id"=>"home", "parent"=>"home", "title"=>"Home",
        "url"=>"index.php" ),
    "contact" => array("id"=>"contact", "parent"=>"home", "title"=>"Contact",
        "url"=>"contact.php" ),
    "services" => array("id"=>"services", "parent"=>"home", "title"=>"Services",
        "url"=>"services.php" ),
    "colpicker" => array("id"=>"colpicker", "parent"=>"services",
    "title"=>"ColorPicker",
        "url"=>"colorpicker.php" ),
    "calendrier" => array("id"=>"calendrier", "parent"=>"services",
    "title"=>"Calendrier",
        "url"=>"calendrier.php" ),
    "dragdroplist" => array("id"=>"dragdroplist", "parent"=>"services",
    "title"=>"DragDropList",
        "url"=>"./dragdroplist/index.html" ),
    "ddstickies" => array("id"=>"ddstickies", "parent"=>"services",
    "title"=>"DropDownStickies",
        "url"=>"./dropdiv/index.php" ),
    "linkgraph" => array("id"=>"linkgraph", "parent"=>"services", "title"=>"LinkGraph",
        "url"=>"./linkgraph/linkgraph.php" ),
    "vectorgraph" => array("id"=>"vectorgraph", "parent"=>"services",
    "title"=>"VectorGraph",
        "url"=>"./vector/index.php" ),
    "slideshow" => array("id"=>"slideshow", "parent"=>"services", "title"=>"Slideshow",
        "url"=>"./slideshow/index.php" )
) ;

foreach ($menus as $menu) {
    $wi = 1 ;
    foreach($menu as $tag => $data) {
        if ($wi == 1) {
            print '      <menu id="' . htmlspecialchars($data) . '">' . PHP_EOL;
            $wi = 2 ;
        }
        print "          <$tag>" . htmlspecialchars($data) . "</{$tag}>" . PHP_EOL;
    }
    print "      </menu>". PHP_EOL;
}
print "</menus>". PHP_EOL;
```

Dans l'exemple ci-dessus, le XML a été créé "manuellement" par concaténation de différentes chaînes de caractères. Cette solution nécessite d'être très vigilant dans l'écriture du code, car il est facile de produire un code XML non conforme (si on oublie d'échapper certains caractères spéciaux). Pour éviter cela, on peut recourir à la classe PHP DomDocument, comme dans l'exemple ci-dessous :

```
$dom = new DomDocument('1.0', 'UTF-8');
$dom->formatOutput = true;

$root = $dom->createElement( "menus" );
$dom->appendChild( $root );

foreach( $menus as $menu )
{
    $bn = $dom->createElement( "menu" );
    $bn->setAttribute( 'id', $menu["id"] );

    $parent = $dom->createElement( "id" );
    $parent->appendChild( $dom->createTextNode( $menu['id'] ) );
    $bn->appendChild( $parent );

    $parent = $dom->createElement( "parent" );
    $parent->appendChild( $dom->createTextNode( $menu['parent'] ) );
    $bn->appendChild( $parent );

    $title = $dom->createElement( "title" );
    $title->appendChild( $dom->createTextNode( $menu['title'] ) );
    $bn->appendChild( $title );

    $title = $dom->createElement( "url" );
    $title->appendChild( $dom->createTextNode( $menu['url'] ) );
    $bn->appendChild( $title );

    $root->appendChild( $bn );
}

header( "Content-type: text/xml" );
echo $dom->saveXML();
```

4.8.3.2 Lire du XML

Pour lire un petit fichier XML, la technique la plus simple consiste à utiliser la fonction PHP `simplexml_load_file()`, comme dans l'exemple suivant :

```
$url = 'menus.xml';
$menus = simplexml_load_file($url);
print '<ul>';
foreach ($menus->menu as $item) {
    print '<li><a href="' . htmlentities($item->id) . '">' .
        htmlentities($item->title) . '</a></li>';
}
print '</ul>';
```

On peut aussi recourir à la classe `DomDocument` :

```
$url = 'menus.xml';

$dom = new DOMDocument;
$dom->load($url);

foreach ($dom->getElementsByTagName('menu') as $menu) {
    // childNodes holds the author values
    $text_nodes = $menu->childNodes;

    $button = array();
    foreach ($text_nodes as $text) {
        $button []= array $text->nodeName => $text->nodeValue) ;
    }
    print_r($button); echo "<br/>" ;
}
```

Pour aider à mieux comprendre le code ci-dessus, voici un échantillon du fichier XML utilisé dans l'exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<menus>
    <menu id="home">
        <id>home</id>
        <parent>home</parent>
        <title>Accueil</title>
        <url>index.php</url>
    </menu>
    <menu id="applications">
        <id>applications</id>
        <parent>home</parent>
```

```

<title>Applications</title>
<url>mnu_applications.php</url>
</menu>
...
</menus>
```

La fonction `simplexml_load_file()` est bien adaptée à la lecture de fichier XML de petite taille, mais la lecture de fichiers XML de grande taille nécessite l'emploi d'un parser SAX, comme dans l'exemple suivant :

Echantillon du fichier XML :

```

<?xml version="1.0" encoding="UTF-8"?>
<books>
<book>
<author>Jack Herrington</author>
<title>PHP Hacks</title>
<publisher>O'Reilly</publisher>
</book>
<book>
<author>Jack Herrington</author>
<title>Podcasting Hacks</title>
<publisher>O'Reilly</publisher>
</book>
</books>
```

Code source PHP utilisant un parser SAX pour la lecture du fichier XML ci-dessus :

```

/*
 * Exemple de Jack Herrington pris sur le site :
 * http://www.ibm.com/developerworksopensource/library/os-xmldomphp/
 */
$g_books = array ();
$g_elem = null;

function startElement($parser, $name, $attrs) {
    global $g_books, $g_elem;
    if ($name == 'BOOK')
        $g_books [] = array ();
    $g_elem = $name;
}

function endElement($parser, $name) {
    global $g_elem;
    $g_elem = null;
}
```

```

function textData($parser, $text) {
    global $g_books, $g_elem;
    if ($g_elem == 'AUTHOR' || $g_elem == 'PUBLISHER' || $g_elem == 'TITLE') {
        $g_books [count ( $g_books ) - 1] [$g_elem] = $text;
    }
}

$paser = xml_parser_create ();

xml_set_element_handler ( $paser, "startElement", "endElement" );
xml_set_character_data_handler ( $paser, "textData" );

$f = fopen ( 'books.xml', 'r' );

while ( $data = fread ( $f, 4096 ) ) {
    xml_parse ( $paser, $data );
}

xml_parser_free ( $paser );

foreach ( $g_books as $book ) {
    echo $book ['TITLE'] . " - " . $book ['AUTHOR'] . " - ";
    echo $book ['PUBLISHER'] . "\n";
}

```

On recommandera la lecture de l'excellent dossier de Jack Herrington sur la manipulation de fichier XML en PHP :

<http://www.ibm.com/developerworks/opensource/library/os-xmldomphp/>

4.8.4 Les fichiers PDF

Des projets PHP existent pour la génération de PDF via PHP, les principaux étant FPDF et TCPDF.

<http://www.fpdf.org>

<http://www.tcpdf.org>

Ces projets sont bien documentés, et s'utilisent sans difficulté, c'est pourquoi ils ne seront pas présentés dans ce support de cours.

A noter l'existence d'une classe FPDI, qui permet d'étendre les possibilités de FPDF, avec notamment la gestion de fond de page, ce qui peut être intéressant pour la production de factures par exemple :

<http://manuals.setasign.com/fpdi-manual/installation/>

4.9 Etudes de cas

Dans ce chapitre, nous allons explorer quelques techniques de développement, au travers de cas concrets directement tirés de projets professionnels. Pour travailler dans de bonnes conditions nous allons commencer par créer un projet, s'appuyant sur le framework CSS Bootstrap, avant d'y intégrer des modules tels que des listes (avec filtres et pagination), ou encore des modules de type CRUD (nous verrons la signification de cet acronyme dans quelques instants).

4.9.1 Préparation d'un projet

Pour la construction de notre projet, nous pouvons utiliser le site « initializr.com » (que nous avions évoqué dans le cours « Bootstrap – Premiers pas »). Il propose un générateur très pratique :

Classic H5BP

Responsive

Bootstrap

Docs Demo Docs Demo Docs Demo

2 - Fine tuning

HTML/CSS Template	HTML5 Polyfills	jQuery
<input type="radio"/> No template	<input type="radio"/> Modernizr	<input checked="" type="checkbox"/> Minified
<input type="radio"/> Mobile-first Responsive	<input checked="" type="radio"/> Just HTML5shiv	<input type="checkbox"/> Development
<input checked="" type="radio"/> Twitter Bootstrap	<input checked="" type="checkbox"/> Respond - Alternatives	

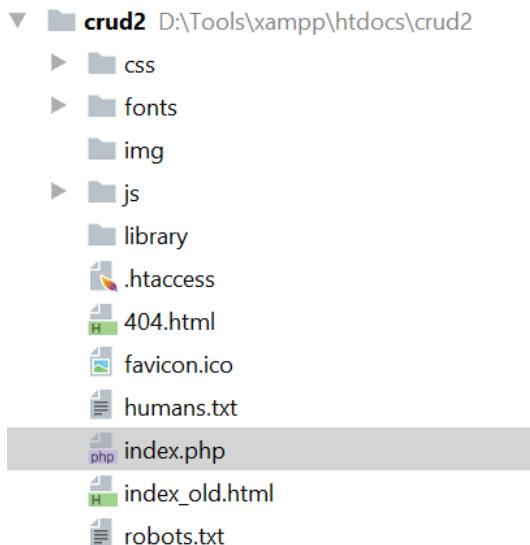
H5BP Optional

<input checked="" type="checkbox"/> IE Classes	<input checked="" type="checkbox"/> Favicon	<input checked="" type="checkbox"/> Humans.txt
<input checked="" type="checkbox"/> Old browser warning	<input type="checkbox"/> Apple and Windows Icons	<input checked="" type="checkbox"/> 404 Page
<input checked="" type="checkbox"/> Google Analytics	<input checked="" type="checkbox"/> plugins.js	<input type="checkbox"/> Adobe Cross Domain
<input checked="" type="checkbox"/> htaccess	<input checked="" type="checkbox"/> Robots.txt	

Download it! **What's inside?**

Nous obtenons un fichier zip dans lequel nous allons récupérer les éléments permettant de démarrer rapidement notre projet. Le projet en lui-même, nous allons le créer dans le répertoire « htdocs » (pour Xamp) ou « www » (si vous travaillez avec un autre stack utilisant ce nom de répertoire). Comme j'avais déjà un projet s'appelant « crud », j'ai décidé de créer ce nouveau projet « crud2 », mais vous pouvez lui donner le nom que vous souhaitez.

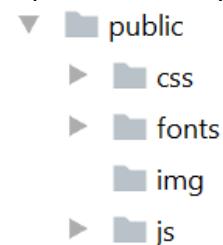
Après avoir copié le contenu du zip fourni par Initializr.com, vous obtenez à peu près ceci :



Une petite différence à noter par rapport au projet fourni par Initializr : j'ai renommé le fichier index.html en index_old.html (car certains éléments de son code pourront nous être utiles et on pourra piocher dedans en cas de besoin). Et j'ai créé un fichier index.php, qui pour l'instant ne contient pas grand-chose (nous le retravaillerons dans un instant) :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>CRUD </title>
</head>
<body>
<?php
// put your code here
?>
</body>
</html>
```

Autre point à noter : j'ai légèrement modifié l'arborescence du projet en ajoutant un répertoire « public » et en y regroupant les répertoires suivants :



Ce n'était bien évidemment pas une obligation, mais il s'agit d'une organisation que l'on retrouve dans de nombreux projets, et je trouve qu'elle est pratique et cohérente, du fait que la

plupart des fichiers CSS, JS, fonts et image sont destinés au navigateur, donc au « public » qui utilise la partie « front » de l'application. Cette organisation des répertoires permet aussi de mieux gérer la sécurité, car on peut appliquer des droits spécifiques plus restrictifs aux fichiers et répertoires qui ne se trouvent pas dans le répertoire « public » (mais c'est en dehors du périmètre de ce cours).

Nous devons aussi créer une base de données MySQL. Pour cela, nous allons nous appuyer sur PhpMyAdmin. Je vous propose d'appeler cette base de données « crudTutorial », et de lui attribuer l'encodage « utf8_general_ci ».



Nous verrons au fil de l'eau quelles tables SQL nous avons besoin de créer.

Nous allons préparer encore quelques éléments qui nous serviront dans la plupart des modules de notre mini-projet.

Je souhaite notamment disposer d'une classe simplifiant la génération les entêtes et pieds de page HTML. Je crée donc une classe HTMLPage, et je la place dans le sous-répertoire « library » (dans un script php « HTMLPage.php »). Voici son code :

```
abstract class HTMLPage {  
  
    public static function getHeader($titre='sans titre') {  
  
        return <<<BLOC_HTML  
<!DOCTYPE html>  
<head lang="fr">  
    <meta charset="utf-8">  
    <title>{$titre}</title>  
    <meta name="description" content="exemple de CRUD">  
    <link rel="stylesheet" href="css/bootstrap.min.css">  
    <style>  
        body {  
            padding-top: 50px;  
            padding-bottom: 20px;  
        }  
    </style>  
    <link rel="stylesheet" href="css/bootstrap-theme.min.css">  
    <link rel="stylesheet" href="css/main.css">
```

```

<!--[if lt IE 9]>
<script src="js/vendor/html5-3.6/respond-1.4.2.min.js"></script>
<![endif]-->
</head>
<body>
BLOC_HTML;
}

public static function getFooter() {
    return <<<BLOC_HTML
<footer class="container">
    <p>PHP - Premiers pas || CRUD Tutorial</p>
</footer>
<script src="js/vendor/jquery-1.11.2.min.js"></script>
<script src="js/vendor/bootstrap.min.js"></script>
<script src="js/plugins.js"></script>
<script src="js/main.js"></script>

</body>
</html>
BLOC_HTML;
}
}

```

Comme il s'agit d'une classe abstraite, nous n'avons pas besoin d'instancier un objet pour l'utiliser. Les méthodes de cette classe étant statiques, la génération d'un entête de page se fait de la façon suivante (avec les double-points au lieu de la flèche) :

```
echo HTMLPage::getHeader('CRUD Exemple 1');
```

... et pour le pied de page, c'est encore plus simple :

```
echo HTMLPage::getFooter();
```

Au fait, Bootstrap nous offre la possibilité de générer de beaux menus déroulants, ce serait dommage de ne pas en profiter. J'ajoute à la classe HTMLPage une méthode getMenu() qui nous permettra d'afficher un squelette de menu Bootstrap

```

public static function getMenu() {
    return <<<BLOC_HTML
<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">GRETA 91 Project</a>
        </div>
    </div>
BLOC_HTML;
}
}

```

```

        <div id="navbar" class="collapse navbar-collapse">
            <ul class="nav navbar-nav">
                <li class="active"><a href="index.php">Accueil</a></li>
                <li class="dropdown">
                    <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-haspopup="true" aria-expanded="false">Travaux <span
class="caret"></span></a>
                    <ul class="dropdown-menu">
                        <li role="separator" class="divider"></li>
                        <li class="dropdown-header">Sous-menu1</li>
                        <li><a href="#">option bidon 1</a></li>
                        <li><a href="#">option bidon 2</a></li>
                        <li role="separator" class="divider"></li>
                        <li class="dropdown-header">Sous-menu2</li>
                        <li><a href="#">option bidon 3</a></li>
                        <li><a href="#">option bidon 4</a></li>
                    </ul>
                </li>
            </ul>
        </div><!-- .nav-collapse -->
    </div>
</nav>
BLOC_HTML;
}

}

```

Pour utiliser cette nouvelle méthode, et faire apparaître le menu, il nous suffira de placer la ligne ci-dessous dans nos scripts :

```
echo HTMLPage::getFooter();
```

Avec un squelette de script comme celui-ci :

```

<?php
require_once 'library/HTMLPage.php';

echo HTMLPage::getHeader('Albums de BD');
echo HTMLPage::getMenu2();

echo '<div class="container">' . PHP_EOL;
echo '<div class="jumbotron">' . PHP_EOL;
echo 'bla bla bla bla';
echo '</div>; //--> end jumbotron

echo '<div class="row">' . PHP_EOL;

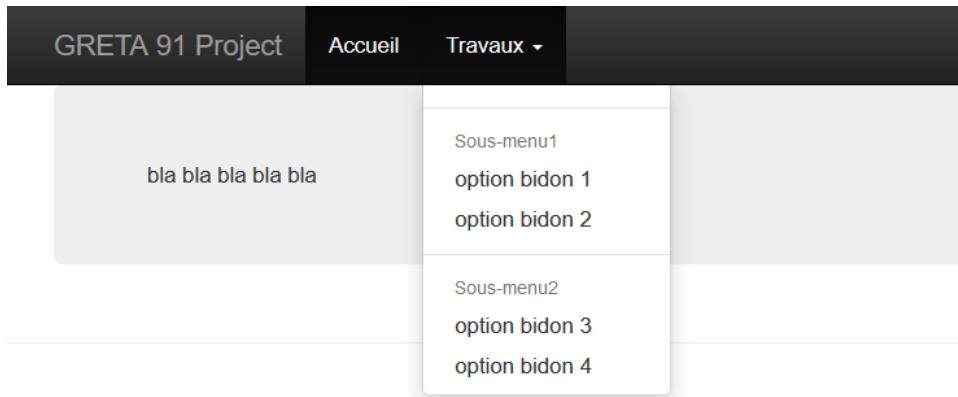
// code métier à placer ici

echo '</div>' . PHP_EOL; //--> end row
echo '</div>' . PHP_EOL; //--> end container

echo HTMLPage::getFooter();

```

... nous sommes en mesure de générer une page avec un menu de ce type :



PHP - Premiers pas || CRUD Tutorial

Nous avons également besoin d'une classe servant à établir la connection avec la base MySQL. Voici son code, placé lui aussi dans le répertoire « library », dans un script PHP portant le même nom que la classe :

```
class DBConnect
{
    private static $dbName = 'crud_tutorial';
    private static $dbHost = 'localhost';
    private static $dbUsername = 'root';
    private static $dbUserPassword = '';

    private static $cont = null;

    public function __construct()
    {
        die('Init function is not allowed');
    }

    public static function open()
    {
        // One connection through whole application
        if ( null == self::$cont )
        {
            try
            {
                $dsn = "mysql:host=".self::$dbHost.";dbname=". self::$dbName;
                self::$cont = new PDO($dsn, self::$dbUsername, self::$dbUserPassword);
                self::$cont->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
                // Passage en mode UTF-8 systématique
                self::$cont->exec("SET NAMES 'utf8'");
            }
            catch(PDOException $e)
            {
                die($e->getMessage());
            }
        }
        return self::$cont;
    }
}
```

```

    }

    public static function close()
    {
        self::$cont = null;
    }
}

```

Cette classe s'appuie sur un « design pattern » (un « modèle de conception ») qui porte le nom de Singleton. C'est quoi la caractéristique du Singleton ? Eh bien, si on appelle plusieurs fois la méthode open() de la classe, cette méthode renverra toujours la même instance de connexion (à la base de données MySQL).

On notera également que cette classe effectue la connexion à la base de données en s'appuyant sur la classe standard PDO, que nous avons déjà présentée au chapitre « PHP et le SQL ».

Pour terminer avec la mise en place des premiers outils, ce serait bien de disposer d'une classe effectuant le nettoyage des champs de formulaire, comme nous l'avons vu dans le chapitre 4.4.4. Je vous propose la classe « Sanitize », qui applique la fonction PHP strip_tags() aux données provenant de \$_POST et \$_GET :

```

<?php

/**
 * Classe destinée à nettoyer les champs de formulaire
 * (de type GET et POST)
 */
class Sanitize {

    /**
     * Sécuriser les champs renvoyés par $_GET
     * @param type $param
     * @return string
     */
    public static function blinder_get($param) {
        if (array_key_exists($param, $_GET)) {
            return self::blindage($_GET[$param]);
        } else {
            return '';
        }
    }

    /**
     * Sécuriser les champs renvoyés par $_POST
     * @param type $param
     * @return string
     */
    public static function blinder_post($param) {
        if (array_key_exists($param, $_POST)) {
            return self::blindage($_POST[$param]);
        } else {
            return '';
        }
    }

    private static function blindage($param) {
        $param = strip_tags($param);
    }
}

```

```

    return trim($param);
}

}

```

4.9.2 Liste avec pagination

La liste avec pagination est un grand classique que tout développeur se doit de connaître.

Rassurez-vous, ce n'est pas très compliqué 😊.

Mais c'est quoi au juste une liste avec pagination ? Vous en avez certainement manipulé sans le savoir. Le plus simple pour vous expliquer, c'est que je vous en montre un exemple.

Liste des pays avec PDO

Critères de sélection

code France	commence par		Valider
CODE FRANCE	CODE ISO	DESCRIPTION	
AF	AFG	AFGHANISTAN	
ZA	ZAF	AFRIQUE DU SUD	
AX	ALA	ALAND, ILES	
AL	ALB	ALBANIE	
DZ	DZA	ALGERIE	
DE	DEU	ALLEMAGNE	
AD	AND	ANDORRE	
AO	AGO	ANGOLA	
AI	AIA	ANGUILLE	
AQ	ATA	ANTARCTIQUE	

<< Préc.	1-10	11-20	21-30	31-40	...	241-246	Suiv. >>
--------------------------------	----------------------	-----------------------	-----------------------	-----------------------	---------------------	-------------------------	--------------------------------

On dit qu'il s'agit d'une liste avec pagination, car il y a cette barre de navigation, sous la liste, qui contient une série de liens cliquables, déterminée en fonction du nombre total d'éléments contenus par la liste à afficher. Dans le cas de notre liste de pays, nous avons 246 éléments à afficher, et comme nous nous sommes volontairement limités à 10 éléments par page, je vous laisse faire le calcul...

A noter que le tableau HTML contenant la liste des pays est « stylée » avec le framework CSS Bootstrap. On utilise en particulier les classes « table », « table-striped » et « table-bordered ».

Pour la barre de navigation, j'ai aussi utilisé des classes fournies par Bootstrap, mais cette partie a nécessité aussi l'ajout d'une classe spécifique, que j'ai créée en m'appuyant sur un exemple fourni dans cet excellent livre :

PHP Cookbook (2nd Edition), de Adam Trachtenberg et David Sklar, O'Reilly (2006)

La classe Pagination que j'ai créée n'est pas la copie exacte de celle proposée par A. Trachtenberg et D. Sklar. En fait, ils proposaient dans leur exemple 2 fonctions que j'ai regroupées au sein d'une classe sous forme de méthodes statiques. J'ai limité le nombre de liens cliquables, car la version initiale proposée dans le livre générait autant de liens que de pages... dans l'exemple de la page précédente, nous aurions abouti à une barre de pagination tenant sur au moins 2 lignes, cela n'aurait pas été vraiment sexy. Plus récemment, j'ai aligné le code généré par ce composant sur les classes CSS fournies par le framework CSS Bootstrap (qui n'existe pas en 2006).

Voici le code source de la classe Pagination :

```
abstract class Pagination {

    public function __construct() {
        throw new Exception("Static class - instances not allowed.");
    }

    static private function pc_print_link($inactive, $text, $offset, $current_page,
                                         $params_page, $bootstrap) {
        // on prépare l'URL avec tous les paramètres sauf "offset"
        if (!isset($offset) or $offset == '' or $offset == '0') {
            $offset = '1';
        }
        $url = '';
        $params_page ['offset'] = $offset;
        $url = '?' . http_build_query($params_page);
        $output = '';
        $current_page = htmlentities($current_page) ;
        if (!$bootstrap) {
            if ($inactive) {
                $output = "<span class='inactive'>$text</span>" . PHP_EOL;
            } else {
                $output = "<span class='active'>" . "<a href='" .
                           $current_page . $url. "'>$text</a></span>" . PHP_EOL;
            }
        } else {
            if ($inactive) {
                $output = "<li class='disabled'><a href='#'$text</a></li>" . PHP_EOL;
            } else {
                $output = "<li class='activex'><a href='" .
                           $current_page . $url. "'>$text</a></li>" . PHP_EOL;
            }
        }
        return $output ;
    }
}
```

```

static private function pc_indexed_links($total, $offset, $per_page, $curpage,
                                         $parmpage, $bootstrap) {
    $separator = ' | ';
    $list_links = [];

    $list_links [] = self::pc_print_link($offset == 1, '<< Pr&eacute;c.', 
                                         $offset - $per_page, $curpage, $parmpage, $bootstrap);

    $compteur = 0;
    $stop_suspension = false;

    // affichage de tous les groupes à l'exception du dernier
    for ($start = 1, $end = $per_page; $end < $total;
          $start += $per_page, $end += $per_page) {
        $compteur += 1;
        if ($compteur < 5) {
            if (!$bootstrap) {
                $list_links [] = $separator;
            }
            $list_links [] = self::pc_print_link($offset == $start, "$start-$end",
                                              $start, $curpage, $parmpage, $bootstrap);
        } else {
            if (!$stop_suspension) {
                $stop_suspension = true;
                if (!$bootstrap) {
                    $list_links [] = ' | ... ';
                } else {
                    $list_links [] = '<li class="disabled"><a href="#"> ... 
</a></li>';
                }
            }
        }
    }

    $end = ($total > $start) ? '-' . $total : '';
}

if (!$bootstrap) {
    $list_links [] = $separator;
}
$list_links [] = self::pc_print_link($offset == $start, "$start$end",
                                    $start, $curpage, $parmpage, $bootstrap);

if (!$bootstrap) {
    $list_links [] = $separator;
}
$list_links [] = self::pc_print_link($offset == $start, 'Suiv. >>',
                                    $offset + $per_page, $curpage, $parmpage, $bootstrap);

return $list_links;
}

static public function pc_indexed_links_text($total, $offset, $per_page,
                                              $curpage, $parmpage) {
    $links = self::pc_indexed_links($total, $offset, $per_page, $curpage,
                                   $parmpage, false);
    return implode(' ', $links);
}

```

```

    static public function pc_indexed_links_bootstrap($total, $offset, $per_page,
                                                    $curpage, $parmpage) {
        $links = self::pc_indexed_links($total, $offset, $per_page, $curpage,
                                         $parmpage, true);
        $html = '<nav><ul class="pagination">' . PHP_EOL;
        $html .= implode('', $links);
        $html .= '</ul></nav>' . PHP_EOL;
        return $html;
    }
}

```

Sympathique cette classe Pagination non ? Bon honnêtement, même si l'adaptation à Bootstrap était intéressante à faire, ce genre de code ne fait pas partie de mes lectures favorites. Je suis content qu'il fonctionne, cela suffit à mon bonheur 😊. Un exemple d'utilisation ? En voici un :

```

$page_en_cours = $_SERVER['PHP_SELF'];
echo Pagination::pc_indexed_links_bootstrap($total, $offset, $nb_per_page,
                                             $page_en_cours, $params);

```

Rassurez-vous, cela vous semblera plus clair, mis en situation avec le code source complet.

De quoi a-t-on encore besoin pour effectuer cette pagination ? Si vous observez la copie d'écran, vous voyez que nous avons deux champs de saisie de type « select » :

Je n'ai pas envie de m'embêter à générer le code HTML de ces deux champs de type « select », alors j'ai récupéré une méthode que j'avais écrite pour un autre projet, et je l'ai collée dans le code de la classe « GenForm », que nous avions créée dans un précédent chapitre. Voici le code de la nouvelle méthode que nous allons utiliser :

```

public function gen_input_select ($nom_champ, $liste_valeurs,
                                $valeur_choisie='') {
    $html = "<select id=\"$id_nom_champ\" name=\"$nom_champ\" class=\"form-control\>".PHP_EOL;
    if (!is_array($liste_valeurs)) {
        error_log('anomalie sur 3ème param. classe:' .
                  __CLASS__ . ' méthode ' .
                  __METHOD__ . ' fichier ' . __FILE__);
    } else {
        foreach ($liste_valeurs as $key=>$value) {
            // utilisation du double égal volontaire, ne pas modifier SVP
            if ($key == $valeur_choisie) {
                $temp_selected = "selected";
            } else {
                $temp_selected = '';
            }
            $html .= "<option value=\"$key\" $temp_selected >$value</option>".PHP_EOL;
        }
    }
    $html .= "</select>".PHP_EOL;
    return $html ;
}

```

Il y a un truc que je ne vous ai pas montré dans la copie d'écran que je vous ai montrée en introduction, c'est que j'ai prévu des mouchards, que j'ai placés sous la barre de pagination, pour connaître le détail des requêtes SQL exécutées :

Liste des pays avec PDO

Critères de sélection

code ISO	commence par	fr	Valider
CODE FRANCE	CODE ISO	DESCRIPTION	
FO	FRO	FEROE, ILES	
FR	FRA	FRANCE	

<< Préc. 1-2 Suiv. >>

Liste des pays dont le code ISO commence par : fr

```

SELECT count(*) AS total FROM countries WHERE codinter LIKE ? ;
SELECT id, codinter, codfra, countryname FROM countries WHERE codinter LIKE ? ORDER BY countryname LIMIT 0, 10;
Valeur du ? : fr%

```

On voit grâce au mouchard qu'il y a 2 requêtes SQL qui sont exécutées à chaque fois que l'on charge une page :

- La première requête est une requête de comptage, elle détermine le nombre total de lignes concernées par la sélection

```
SELECT count(*) AS total FROM countries WHERE codinter LIKE ? ;
```

- La seconde requête charge la liste des pays, et effectue la pagination dans le même temps :

```
SELECT id, codinter, codfra, countryname
FROM countries
WHERE codinter
LIKE ?
ORDER BY countryname LIMIT 0, 10;
```

Les paramètres de la clause SQL « limit » évolue - pour chaque nouvelle page sélectionnée - en fonction du lien sélectionné dans la barre de pagination :

The screenshot shows a user interface for a database search. At the top, there is a navigation bar with links: '<< Préc.', '1-10', '11-20', '21-30' (which is highlighted with a red box and has a red arrow pointing down to it), and 'Suiv. >>'. Below this is a green header bar containing the text 'Liste des pays dont le libellé contient : ar'. At the bottom, there is a grey box containing two SQL queries and a placeholder value:

```
SELECT count(*) AS total FROM countries WHERE countryname LIKE ? ;
SELECT id, codinter, codfra, countryname FROM countries WHERE countryname LIKE ? ORDER BY countryname LIMIT 20, 10;
Valeur du ? : %ar%
```

Vous aurez sans doute remarqué que les 2 requêtes SQL sont des requêtes SQL paramétrées. On le reconnaît à la présence du point d'interrogation. Pour des raisons de sécurité - déjà évoquées dans le chapitre d'introduction à l'utilisation de SQL - nous ne « collerons » jamais par concaténation dans une requête SQL, des informations provenant d'un formulaire. Ce serait trop dangereux, car cela nous exposerait au risque d'attaque par « injection SQL ».

Je me dois d'insister sur le fait que l'utilisation d'une requête SQL paramétrée nous protège des attaques par injection SQL, dans la mesure où c'est le moteur SQL qui va se charger d'analyser et d'exploiter la donnée qui lui est transmise (et la plupart des moteurs SQL disposent d'algorithmes bien bétonnés pour ça).

Donc, il y a deux requêtes SQL à traiter dans notre affaire... Très franchement, j'aime beaucoup PDO, mais toute la mécanique qu'il faut écrire pour récupérer les lignes et colonnes renvoyées

par une requête SQL me barbe prodigieusement. Si je vous propose les 2 solutions ci-dessous, vous préférez laquelle, la première ou la seconde ?

- 1^{ère} solution :

```
try {
    $st2 = $db->prepare($requete2b);
    $ok = $st2->execute($where_pdo);
    if ($ok) {
        $row = $st2->fetch(PDO::FETCH_ASSOC);
        while ($row != false) {
            echo <<<LOTABLE
</tr>
<td width="15%">{$row['codfra']}</td>
<td width="15%">{$row['codinter']}</td>
<td width="70%">{$row['countryname']}</td>
</tr>
LOTABLE;
            $row = $st2->fetch(PDO::FETCH_ASSOC);
        }
    }
    unset($st2);
} catch (PDOException $e) {
    // envoi vers la log PHP des différents message produits par la classe PDOException
    error_log( "getMessage -> " . $e->getMessage() );
    error_log( "getCode -> " . $e->getCode() );
    error_log( "getFile -> " . $e->getFile() );
    error_log( "getLine -> " . $e->getLine() );
    error_log( "getTrace -> " . $e->getTraceAsString() );
}
```

- 2^{ème} solution :

```
$datas = DBWrapper::selectBlock($db, $requete2b, $where_pdo);
foreach($datas as $data) {
    echo <<<TABLEROW
</tr>
<td width="15%">{$data['codfra']}</td>
<td width="15%">{$data['codinter']}</td>
<td width="70%">{$data['countryname']}</td>
</tr>
TABLEROW;
}
```

La 2^{ème} solution, c'est une classe DBWrapper, qui encapsule la mécanique propre à PDO. Dans notre exemple, nous utilisons la méthode « selectBlock », à laquelle nous passons 3 paramètres :

- Un objet PDO (initialisé en amont),
- la requête SQL à exécuter
- le tableau des paramètres de la requête.

Si une erreur se produit pendant l'exécution de la requête, la méthode selectBlock() se charge d'envoyer le détail de l'erreur dans la log de PHP... Si au contraire tout s'est bien passé, alors la méthode selectBlock() renvoie un tableau contenant les données à afficher.

Ce composant DBWrapper m'avait été inspiré par un exemple trouvé dans l'excellent bouquin que voici :

PHP Hacks, de Jack D. Herrington (éditions O'Reilly, 2009)

Quoiqu'un peu ancien, ce livre fourmille de bonnes idées et bons conseils, encore très pertinents aujourd'hui, je vous en recommande la lecture.

A noter que le composant DBWrapper que je vous ai donné dans ce support est une version « lite » d'un composant que j'avais créé pour répondre à mes besoins de développements, notamment pour la base de données DB2 d'IBM. J'avais appelé ce composant MacaronDB, il est disponible sur ce dépôt :

<https://github.com/gregja/macarondb>

Vous pouvez l'utiliser, ou éventuellement vous en inspirer.

Bien que plus allégé que le composant MacaronDB, le code de la classe DBWrapper que nous utilisons dans ce cours est un peu volumineux, je l'ai donc placé dans un chapitre dédié en annexe.

Je crois que nous avons fait le tour du sujet, il ne nous reste plus qu'à lire le code source du script « pdo_listepays.php » :

```
<?php
require_once 'library/GenForm.php';
require_once 'library/HTMLPage.php';
require_once 'library/Sanitize.php';
require_once 'library/Pagination.php';
require_once 'library/DBConnect.php';
require_once 'library/DBWrapper.php';

echo HTMLPage::getHeader('Liste des pays');
echo HTMLPage::getMenu();

echo '<div class="container">' . PHP_EOL ;
echo '<h2>Liste des pays avec PDO</h2>', PHP_EOL;

echo '<form id="listpays" method="get">', PHP_EOL;
echo '<fieldset>', PHP_EOL;
echo '<legend>Crit&egrave;res de s&acute;lection</legend>', PHP_EOL;
```

```

$liste_cols = array(
    1 => 'code France',
    2 => 'code ISO',
    3 => 'libellé'
);

$form = new GenForm();
echo '<label>';
echo $form->gen_input_select('col_rech', $liste_cols,
Sanitize:::blinder_get('col_rech'));
echo '</label>';

$type_rech = array(
    1 => 'égal',
    2 => 'contient',
    3 => 'commence par'
);
echo ' ';
echo '<label>';
echo $form->gen_input_select('typ_rech', $type_rech,
Sanitize:::blinder_get('typ_rech'));
echo '</label>';
echo ' ';
echo '<label>';
echo $form->gen_input_text('lib_rech', Sanitize:::blinder_get('lib_rech'));
echo '</label>';
echo ' ';
echo '<input type="submit" value="Valider" class="btn btn-success" />';
echo '</form>';

echo '<br/>', PHP_EOL;

if (count($_GET) > 0) {

    // création d'une instance de PDO
    $db = DBConnect:::open();

    // alternance de classes bootstrap pour les lignes du tableau HTML
    $styles_tab = array('info', 'active');

    // récupération des paramètres de $_GET
    $params = array();
    $params ['col_rech'] = Sanitize:::blinder_get('col_rech');
    $params ['typ_rech'] = Sanitize:::blinder_get('typ_rech');
    $params ['lib_rech'] = Sanitize:::blinder_get('lib_rech');
    if ($params ['col_rech'] == '' ||
        $params ['typ_rech'] == '' ||
        count($_GET) == 0) {
        // si $_GET contient des données illicites,
        // alors redirection sur page d'accueil
        header("Location: ./index.php");exit;
    }

    // récupération ou initialisation de l'offset
    $offset = isset($_GET ['offset']) ? intval($_GET ['offset']) : 1;
    if ($offset == 0) {
        $offset = 1;
    }

    $lib_titre = " Liste des pays ";
}

```

```

// Préparation des requêtes 1 et 2
// La requête 1 sert au comptage du nombre d'occurrences
$requete1 = "SELECT count(*) AS total FROM countries";

// La requête 2 sert au chargement de la liste des pays
$requete2 = 'SELECT id, codinter, codfra, countryname FROM countries';

// Tableau contenant le critère de filtrage
$where_pdo = array();

// Récupération du critère de recherche par libellé
if ($params ['lib_rech'] != '') {
    $lib_titre .= ' dont le ';
    switch ($params ['col_rech']) {
        case '1' : {
            $lib_titre .= ' code France ';
            $where_sql = ' codfra ';
            break;
        }
        case '2' : {
            $lib_titre .= ' code ISO ';
            $where_sql = ' codinter ';
            break;
        }
        case '3' : {
            $lib_titre .= ' libellé';
            $where_sql = ' countryname ';
            break;
        }
        default : {
            $msg_erreur = 'Critères de recherche incorrects';
            error_log($msg_erreur);
            $params ['col_rech'] = '1';
        }
    }
}
$type_rech = $params ['typ_rech'];
switch ($type_rech) {
    case '1' : {
        // recherche de type "égal"
        $type_recherche = " = ? ";
        $lib_titre .= " est égal à : " .
            htmlentities(trim($params ['lib_rech']));
        $where_pdo [] = $params ['lib_rech'];
        break;
    }
    case '2' : {
        // recherche de type "contient"
        $type_recherche = " LIKE ? ";
        $lib_titre .= " contient : " . htmlentities($params ['lib_rech']);
        $where_pdo [] = '%' . trim($params ['lib_rech']) . '%';
        break;
    }
    case '3' : {
        // recherche de type "commence par"
        $type_recherche = " LIKE ? ";
        $lib_titre .= " commence par : " .
            htmlentities($params ['lib_rech']);
        $where_pdo [] = trim($params ['lib_rech']) . '%';
        break;
    }
}

```

```

        }
    default : {
        $msg_erreur = 'Crit&egrave;res de recherche incorrects';
        error_log($msg_erreur);
    }
}
$requete_compl = " WHERE " . $where_sql . $type_recherche;
$requete1 .= $requete_compl;
$requete2 .= $requete_compl . ' ORDER BY countryname';
}

// Nombre de lignes maximum par page
$nbl_par_page = 10;

// Comptage du nombre de lignes total pour le critère de recherche
// considéré (en vue de calculer le nombre de pages total)
$total = 0;

$data = DBWrapper::selectOne($db, $requete1, $where_pdo);
$total = isset($data['total'])?(int)$data['total']:0;

if (($offset + $nbl_par_page) > $total) {
    $tmp_offset2 = $total - $offset + 1;
} else {
    $tmp_offset2 = $nbl_par_page;
}
$offset_sql = $offset - 1;
// Ajout de la clause "limit" pour la pagination
$requete2b = $requete2 . " LIMIT $offset_sql, $nbl_par_page";

echo <<<BOTABLE
<table class="table table-striped table-bordered"
summary = "{$lib_titre}" >
<thead>
<tr>
<th>CODE FRANCE</th>
<th>CODE ISO</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
BOTABLE;

$datas = DBWrapper::selectBlock($db, $requete2b, $where_pdo);
foreach($datas as $data) {
    echo <<<TABLEROW
<tr>
    <td width="15%">{$data['codfra']}</td>
    <td width="15%">{$data['codinter']}</td>
    <td width="70%">{$data['countryname']}</td>
</tr>
TABLEROW;
}

echo <<<EOTABLE
</tbody>
</table>
EOTABLE;

// Affichage de la barre de pagination

```

```

$page_en_cours = $_SERVER ['PHP_SELF'];
echo Pagination:::pc_indexed_links_bootstrap($total, $offset, $nbl_par_page,
    $page_en_cours, $params);

// Détail des requêtes SQL exécutées
echo "<div class=\"alert alert-success\">{$lib_titre}</div>";
echo '<pre>';
echo $requete1.';';
echo '<br>';
echo $requete2b.';';
echo '<br>';
if (count($where_pdo)>0) {
    echo 'Valeur du ? : ' . $where_pdo[0];
}
echo '</pre>';

echo '</div>' . PHP_EOL;

echo HTMLPage:::getFooter();

```

Le code que je viens de vous fournir appelle diverses remarques. Comme il s'agit de sujets très différents, je vais les répartir dans différents sous-chapitres.

4.9.2.1 Comptage à la façon d'« Inception »

Le code que je viens de vous montrer pourrait être amélioré à différents niveaux. Mais j'ai préféré laisser ce code « dans son jus » car l'objectif ici n'est pas de vous montrer un code super optimisé, mais plutôt de vous aider à comprendre les mécanismes en jeu dans une pagination avec filtre.

Vous remarquerez que le plus gros du code est occupé finalement par l'analyse des données du formulaire, et leur utilisation pour produire les clauses « WHERE » des 2 requêtes SQL.

La première requête SQL, celle qui sert à effectuer le comptage du nombre de lignes total, se présente ainsi :

```
SELECT count(*) AS total FROM countries
```

Cette requête nécessite quelques précisions.

Dans notre cas, nous sommes sur une requête SQL très simple, mais il arrive que l'on soit amené à écrire des requêtes plus complexes, impliquant plusieurs tables liées par des jointures (nous en verrons un exemple au chapitre suivant). Dans ce cas le comptage avec un **SELECT count(*)** ne pourra se faire sous la forme ci-dessus. Nous devrons feinter en transformant notre requête principale en sous-requête, et en effectuant le comptage sur le résultat renvoyé par la sous-requête, comme ceci :

```
SELECT count(*) FROM (
    SELECT codinter, codfra, countryname FROM countries
) XXX
```

Vous voyez que la requête principale, celle qui renvoie les données « métier » est encapsulée dans des parenthèses, à l'intérieur d'une clause FROM appartenant à une requête de niveau supérieur (qui elle est spécialisée dans le comptage des données). Si vous n'êtes pas habitué à cette manière de faire, sachez qu'elle est assez courante et qu'elle ne sert pas qu'à gérer des problèmes de pagination.

Imaginez que cela fonctionne un peu comme dans le film « Inception » de Christopher Nolan, avec cette histoire de rêves imbriqués : la sous-requête s'exécute, génère un jeu de lignes et colonnes qu'elle renvoie à la requête de niveau supérieur. Cette dernière était en attente du résultat, dès qu'elle le reçoit, elle s'exécute, analyse le jeu de données qu'elle a reçu via la clause FROM, et procède – dans notre exemple – au comptage du nombre de lignes.

Vous noterez la présence de « XXX » après la parenthèse, il s'agit d'un alias que nous avons attribué à la sous-requête. J'aurais pu l'appeler « temp », peu importe le nom, l'important c'est de ne pas l'oublier, car sans la présence de cet alias, la sous-requête ne pourra pas s'exécuter.

Si vous travaillez avec d'autres SGBD que MySQL, vous rencontrerez quelquefois l'écriture suivante, qui dans notre cas est strictement équivalente à la requête précédente :

```
WITH XXX AS (
    SELECT codinter, codfra, countryname FROM countries
)
SELECT count(*) from XXX
```

Il s'agit d'une CTE (Common Table Expression), une technique qui fonctionne sur PostgreSQL et DB2, mais pas sur MySQL. Je la trouve très pratique, mais il est vrai que dans notre exemple, l'avantage de cette technique n'est pas flagrant. Je vous demande de faire un effort d'imagination, et d'envisager une sous-requête plus complexe composée de plusieurs lignes de code.

4.9.2.2 Question de durée

Il faut savoir que vos scripts PHP ont une durée maximale d'exécution qui est définie dans le fichier php.ini de votre stack PHP (cf. extrait ci-dessous) :

```
;;;;;;;;;;;  
; Resource Limits ;  
;;;;;;;;;;;  
  
; Maximum execution time of each script, in seconds  
; http://php.net/max-execution-time  
; Note: This directive is hardcoded to 0 for the CLI SAPI  
max_execution_time=30
```

Eh oui, en règle générale, cette limite est fixée à 30 secondes. Mais comme vous avez accès à ce fichier de configuration, vous pouvez éventuellement passer cette limite à 60 secondes. Mais dites-vous bien que 60 secondes, c'est beaucoup pour un utilisateur, et qu'il risque de quitter prématièrement votre site s'il constate que les pages mettent trop de temps à se charger. Donc prudence.

Si l'utilisateur de votre site sait qu'il utilise une application interne à l'entreprise et que cette application gère une forte volumétrie de données, il sera peut être enclin à être plus patient que la moyenne. Si vous êtes dans ce contexte, prenez le temps de discuter avec lui de son ressenti par rapport à la rapidité de l'application.

Si vous préférez ne pas modifier le fichier php.ini (ou si vous n'avez pas les droits suffisants pour le faire), vous pouvez ponctuellement augmenter la durée maximale d'un script en utilisant la fonction PHP suivante :

```
set_time_limit();
```

Soyez prudent dans l'utilisation de cette fonction, n'en abusez pas, et un conseil, prenez le temps de lire sa documentation sur php.net, car elle a quelques limitations.

Cette problématique de durée, on la retrouve dans le chapitre qui suit, car très souvent, quand un script PHP met beaucoup de temps à renvoyer un résultat, c'est qu'il est lui-même en attente de données renvoyées par SQL.

4.9.2.3 Question de volumétrie

Dans le cas de notre table des pays, le volume de données est restreint, la requête va donc s'exécuter en un temps record, avec ou sans indexs. Mais il arrivera que vous soyez confronté à des volumes de données nettement plus importants.

Si la requête traite un très gros volume de données (de l'ordre de plusieurs centaines de milliers de lignes, voire beaucoup plus...), il convient de se poser quelques questions, notamment sur la pertinence de notre module de consultation avec pagination.

Car même avec la présence d'indexs adaptés, nos 2 requêtes SQL vont nécessiter un temps d'exécution conséquent. Du coup, il n'est peut être pas pertinent de laisser un utilisateur lancer ce genre de requêtes de manière interactive.

On peut éventuellement proposer à notre utilisateur de lancer le traitement de manière asynchrone, et de lui envoyer le résultat sous la forme d'un fichier CSV ou Excel, une fois le résultat connu (sous réserve bien sûr que la taille du fichier résultat soit raisonnable).

Pour la soumission de traitement asynchrone en PHP, des solutions existent, comme le projet RabbitMQ. Pour la génération de fichier CSV, ce sujet est traité dans ce support de cours, mais si votre utilisateur préfère un fichier Excel, des solutions existent comme le projet PHPExcel.

4.9.2.4 Solutions concurrentes et leurs effets de bord

Beaucoup de développeurs PHP ne connaissent pas la technique de pagination que j'ai présentée dans ce chapitre. Mais alors, comment font-il pour gérer des paginations ? Eh bien, ils se rabattent généralement sur des projets Javascript comme Datatables ou JQGrid. Il s'agit de projets écrits en Javascript, qui sont très puissants et très facile à mettre en œuvre. Avec ce que vous avez appris dans le cours « Javascript – Premiers pas » et en lisant la documentation officielle de ces projets, vous n'aurez aucun mal à les mettre en œuvre.

Le problème, c'est qu'il s'agit de projets Javascript, donc de mécanismes qui s'exécutent du côté du client (du navigateur). Cela signifie que les développeurs PHP qui utilisent ces outils génèrent un tableau HTML contenant l'intégralité du jeu de données sur lequel la pagination va être mise en œuvre. Dans notre exemple, cela consisterait à générer un tableau HTML contenant les 246 pays de notre table SQL. Ce tableau HTML va être envoyé dans son intégralité au navigateur de l'internaute. Et c'est le projet Datatables (ou son concurrent JQGrid) qui va prendre le relais pour afficher le tableau sous forme de plusieurs pages, et générer la barre de navigation correspondante.

Des outils comme Datatables et JQGrid sont tellement pratiques et faciles à mettre en œuvre que les développeurs PHP ont tendance à en abuser. Or il y a des effets pervers induits par l'utilisation de ces outils, je m'explique :

- puisqu'il n'y a pas de pagination côté serveur, le serveur SQL est obligé de lire la totalité des données et de les renvoyer à l'interpréteur PHP. La communication entre le moteur SQL et l'interpréteur PHP est gourmande en ressources, le processeur va chauffer excessivement, à la fois pour l'exécution du code SQL, et pour le transfert des données vers PHP. De la mémoire va être consommée de manière excessive pendant ce processus (même si c'est temporaire). L'interpréteur PHP va devoir générer la totalité du tableau HTML, encore du travail dont le processeur du serveur se serait bien passé.
- La transmission d'un tableau HTML complet sur le réseau internet fait « ramer » excessivement tous les serveurs et toutes les passerelles par lesquels la requête HTTP passe avant d'arriver sur le client final.
- Le client final (qui est peut être un smartphone avec peu de ressources) se retrouve obligé de charger le code Javascript de Datatables (ou de son concurrent), puis de charger la totalité du tableau HTML, alors que l'utilisateur du terminal ne sera probablement pas intéressé par l'intégralité du jeu de données qu'il a reçu. Le terminal (smartphone ou autre) va être obligé d'exécuter le code Javascript qui va gérer le mécanisme de pagination. S'il s'agit d'un smartphone avec peu de mémoire, ce travail va le faire « ramer », et consommer de la batterie, donc réduire l'autonomie du smartphone.

Il faut bien comprendre que, toutes les opérations inutiles que l'on demande au serveur, au client, et à toutes les machines intermédiaires sur le réseau internet, sont génératrices de CO₂. Et de surcroît, cela se traduit pour les entreprises par des factures d'électricité anormalement élevées.

4.9.3 Sous-requêtes scalaires, données temporelles

Le chapitre précédent nous a permis de nous familiariser avec la pagination. Notre liste de pays était un cas très simple, dans lequel la requête SQL s'appuyait sur une seule table. Pas besoin dans ce cas d'effectuer de jointures entre plusieurs tables. Mais dans la « vraie vie », on rencontre rarement des cas aussi simples, les requêtes SQL sont souvent plus complexes que cela.

Je vous propose une petite étude de cas, largement inspirée d'un projet réel sur lequel j'ai travaillé il y a quelques années. Le projet en question consistait à développer une gestion de catalogue produit, pour une société spécialisée dans la revente de matériel agricole. L'activité de la société en réalité importe peu, car les techniques que nous allons étudier sont valables pour tout type d'activité, mais cela explique pourquoi le jeu d'essai que nous allons utiliser parle de motoculteur, de tronçonneuse, etc... Je l'ai bien évidemment dépersonnalisé, et largement simplifié, en ne gardant que les tables et colonnes qui me semblaient utiles pour notre étude de cas.

Un point très important également : les techniques SQL que je vais présenter dans ce chapitre fonctionnent sur de nombreuses bases de données. Je les ai pour ma part utilisées avec succès sur les SGBD (Systèmes de Gestion de Base de Données) MySQL, PostgreSQL et DB2. Je ne doute pas que vous pourrez les utiliser sur d'autres SGBD sans trop de modifications.

Nous n'allons pas développer ici toute la gestion de catalogue produit, nous allons plutôt développer un module de consultation proche de celui présenté au chapitre précédent, mais avec des requêtes SQL plus complexes, car les données que nous souhaitons afficher sont réparties dans plusieurs tables.

Dans notre étude de cas, nous disposons de 4 tables :

- prd_produit : la table des produits, comme son nom l'indique
- prd_famille : la table des familles de produits
- prd_activite : la table des activités
- prd_prixvte : la table des prix de vente

Nous avons quelques règles fonctionnelles à énoncer :

- chaque produit est lié à une famille et une seule, mais une famille peut être associée à 0 ou N produits
- chaque famille est liée à une activité et une seule, mais une activité peut être associée à 0 ou N familles

- chaque prix de vente est lié à un produit et un seul
- un produit peut avoir zéro ou N prix de vente, mais il ne peut avoir qu'un seul prix de vente à une date donnée

Prenons pour exemple la table des activités. Voici sa structure :

```

1 CREATE TABLE prd_activite (
2   id int(10) unsigned NOT NULL AUTO_INCREMENT,
3   nom varchar(80) DEFAULT NULL,
4   PRIMARY KEY (id)
5 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 ;|
```

... suivi de son jeu de données :

```

INSERT INTO prd_activite (id, nom) VALUES
(1, 'Motoculture'),
(2, 'Remorques'),
(3, 'Quads'),
(4, 'Cycles'),
(5, 'Matériel agricole et espace vert'),
(6, 'Matériel de travaux publics');
```

Il s'agit d'une table vraiment très simple, avec 2 colonnes qui contiennent respectivement :

- un identifiant numérique en incrémentation automatique,
- un nom de 80 caractères de long (maxi),

Je vous disais que les techniques SQL que nous allions utiliser étaient quasiment identiques sur plusieurs SGBD, et c'est vrai. Mais dans la norme SQL, il y a 2 grandes parties qui sont :

- DDL (Data Definition Language) : cela correspond au code SQL de création et de gestion des tables et autres objets SQL (vues, procédures stockées, etc...)
- DML (Data Manipulation Language) : cela désigne le code SQL d'interrogation et de manipulation du **contenu** des tables, soit les requêtes SELECT, UPDATE, INSERT et DELETE, et toutes les techniques de jointure qui vont avec.

Les plus grosses différences syntaxiques entre SGBD se situent surtout au niveau de DDL, mais on trouve aussi des différences côté DML, notamment sur les fonctions (de conversion de type, de manipulation de dates, etc...), et aussi sur certaines techniques d'interrogation avancées qui ne fonctionnent que sur certains SGBD. Je pense par exemples aux techniques de type [CTE](#)

(Common Table Expression) qui fonctionnent à peu près de la même manière sur PostgreSQL et DB2, et n'existent tout simplement pas sur MySQL.

Jetons maintenant un coup d'œil à la structure des tables familles, produits et prix de vente.

```

1 CREATE TABLE prd_famille (
2     id int(10) unsigned NOT NULL AUTO_INCREMENT,
3     nom varchar(80) DEFAULT NULL,
4     prd_activite_id int(10) unsigned NOT NULL,
5     PRIMARY KEY (id)
6 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 ;

1 CREATE TABLE prd_produit (
2     id int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
3     codeproduit char(50) NOT NULL,
4     prd_famille_id int(10) UNSIGNED NOT NULL,
5     PRIMARY KEY (id)
6 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1 CREATE TABLE prd_prixstd (
2     id int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
3     prd_produit_id int(10) UNSIGNED NOT NULL,
4     date_deb date NOT NULL,
5     prix decimal(11,5) NOT NULL,
6     PRIMARY KEY (id)
7 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

Comme les jeux de données sont un peu plus volumineux pour ces tables, je les ai regroupés dans un chapitre de l'annexe (cf. chapitre 5.8).

Vous avez remarqué que chaque table a une colonne « id ». Cette colonne « id » est un identifiant numérique en incrémentation automatique, qui sert de clé primaire (cf. mot clé « PRIMARY KEY »). Chaque ligne d'une table a donc un identifiant unique qui la définit. Dans certaines applications professionnelles, il peut arriver que la clé primaire d'une table soit composée de plusieurs colonnes, on parle alors de « clé primaire composite ». Ce ne sera pas le cas dans notre application, je n'ai pas souhaité la compliquer plus que nécessaire.

Nous avons précisé tout à l'heure qu'une famille était liée à une activité et une seule. Techniquement cela se traduit par la présence d'une colonne « prd_activite_id » à l'intérieur de la table « prd_famille ». On dit que « prd_activite_id » est une « clé étrangère ».

Pour obtenir la liste des familles avec leurs activités respectives, nous pouvons écrire une jointure entre les tables « famille » et « activité », avec le code SQL suivant :

```
SELECT a.id, a.nom, a.prd_activite_id, b.nom
FROM prd_famille a
INNER JOIN prd_activite b
    ON a.prd_activite_id = b.id
ORDER BY b.id, a.id ;
```

Voici un petit échantillon de ce que produit la requête :

id	nom	prd_activite_id	nom
26	Tracteurs de pelouses	1	Motoculture
27	Tronçonneuses à chaîne	1	Motoculture
28	Tronçonneuses d'élagage	1	Motoculture
29	Tronçonneuses sur perche	1	Motoculture
56	Pulvérisateur	1	Motoculture
61	Tondeuses robot	1	Motoculture
30	Accessoires	2	Remorques
31	Agricoles	2	Remorques
34	Utilitaires	2	Remorques
36	Sports (homologués route)	3	Quads
37	Autres	4	Autres

La clause « INNER JOIN » permet de sélectionner dans la jointure uniquement les lignes qui existent dans les 2 tables liées par la jointure. Si certaines familles sont liées à des activités qui n'existent pas (pour cause de suppression accidentelle), alors elles n'apparaîtront pas avec ce type de requête.

Des familles qui « pointent » sur des activités qui n'existent pas, on appelle ça des orphelins (ou des « lignes orphelines »), en jargon de professionnel (je vous garantis que ce n'est pas une blague).

Vous pourriez être tenté de me répondre que les orphelins ne pourraient pas exister si des contraintes d'intégrité référentielle avaient été définies au niveau SQL, et je vous répondrai que vous avez raison. Mais les contraintes d'intégrité référentielles ne sont que très rarement mises en œuvre dans les projets en entreprise, pour diverses raisons sur lesquelles je ne m'étendrai pas ici. Donc, dans la « vraie vie », des bases de données avec des données orphelines, je pense que vous allez en rencontrer souvent.

Si vous pensez que vous risquez d'avoir des familles orphelines de leurs activités, et si vous souhaitez les voir apparaître quand même, vous pouvez modifier légèrement la requête SQL en remplaçant la clause « INNER JOIN » par la clause « LEFT JOIN », comme ceci :

```
SELECT a.id, a.nom, a.prd_activite_id, b.nom
FROM prd_famille a
LEFT JOIN prd_activite b
    ON a.prd_activite_id = b.id
ORDER BY b.id, a.id ;
```

Il y a une autre manière de lier les 2 tables famille et activité, c'est en utilisant une sous-requête scalaire. Cette technique est moins connue que la précédente, et dans certains cas elle est bien pratique, alors je vous montre ce que cela donne :

```
SELECT a.id, a.nom, a.prd_activite_id ,
(SELECT b.nom FROM prd_activite b
 WHERE a.prd_activite_id = b.id) as nom_activite
FROM `prd_famille` a
ORDER BY nom_activite, a.id
```

On voit dans notre exemple que la sous-requête est définie dans la clause SELECT de la requête principale. Je l'ai « stabilisée » pour vous aider à la repérer, elle est encapsulée dans un jeu de parenthèses, et elle alimente le contenu d'une colonne générée dynamiquement qui s'appelle « nom_activite ».

La sous-requête scalaire a pour particularité de renvoyer une seule ligne et une seule colonne. S'il y a un risque que la sous-requête scalaire renvoie plus d'une ligne, on peut l'obliger à n'en renvoyer qu'une seule en utilisant la clause « limit 1 » comme dans l'exemple suivant :

```
SELECT a.id, a.nom, a.prd_activite_id ,
(SELECT b.nom FROM prd_activite b
 WHERE a.prd_activite_id = b.id LIMIT 1) as nom_activite
FROM `prd_famille` a
ORDER BY nom_activite, a.id
```

Bon, dans notre exemple, la clause « LIMIT 1 » est inutile (de par la nature de notre jeu d'essai), mais vous pourrez trouver d'autres cas où cette technique peut rendre service.

Nous avons utilisé la sous-requête scalaire à l'intérieur de la clause SELECT, mais on peut l'utiliser pour récupérer une information que l'on souhaite comparer avec une donnée de la requête principale, et dans ce cas on placera la sous-requête scalaire à l'intérieur de la clause WHERE. Nous verrons une mise en application dans un instant.

Intéressons-nous à la table des produits, et voyons comment nous pouvons lier ensemble différentes tables. Si je souhaite obtenir la liste des produits, triés par famille et code produit, avec en affichage les libellés activité et famille, suivi du code produit, je peux écrire ceci :

```

SELECT a.nom as nom_famille,
(SELECT b.nom FROM prd_activite b
 WHERE a.prd_activite_id = b.id LIMIT 1) as nom_activite,
c.codeproduit as code_produit
FROM `prd_famille` a
INNER JOIN prd_produit c
ON a.id = c.prd_famille_id
ORDER BY a.nom, c.codeproduit
;

```

Voici un petit échantillon du résultat obtenu :

nom_famille	nom_activite	code_produit
Aspirateurs électriques	Motoculture	SE 61
Aspirateurs électriques	Motoculture	SE 61 E
Broyeurs	Motoculture	GB 370
Broyeurs	Motoculture	GB 370 S
Broyeurs	Motoculture	GE 103
Broyeurs	Motoculture	GE 105
Broyeurs	Motoculture	GE 150
Broyeurs	Motoculture	GE 250
Broyeurs	Motoculture	GE 35 L
Débroussailleuses	Motoculture	FS 100
Débroussailleuses	Motoculture	FS 100 R
Débroussailleuses	Motoculture	FS 130

Et si je veux afficher les tarifs de chaque produit, je peux faire pareil avec une jointure entre la table des produits et la table des prix de vente ?

Un ange passe... suivi d'un petit sourire énigmatique du prof (certains élèves diraient « petit sourire sadique du prof »). Les élèves commencent à se trémousser sur leurs chaises, en se disant qu'il doit y avoir un piège... car il y en a un, forcément !! 😊

Bon, pour expliquer le fond du problème, je vous propose de partir d'une requête toute simple liant uniquement les tables produit et prix de vente (les libellés famille et activité, laissons-les de côté, on les rajoutera plus tard).

Commençons par une première jointure :

```

SELECT a.codeproduit as code_produit,
a.id as produit_id,
b.prd_produit_id, b.date_deb, b.prix
FROM prd_produit a
LEFT JOIN prd_prixstd b
ON a.id = b.prd_produit_id
ORDER BY a.codeproduit, b.date_deb
;

```

Un petit échantillon du résultat produit :

code_produit	produit_id	prd_produit_id	date_deb	prix
BR 500	80	80	2017-01-01	739.97000
BR 550	81	81	2017-01-01	756.69000
BR 600	82	82	2017-01-01	848.66000
BT 121 C	83	NULL	NULL	NULL
BT 45 Perceuse à bois	31	NULL	NULL	NULL
BT 45 Tarière	27	27	2017-01-01	581.10000
FR 130 T	121	NULL	NULL	NULL
FR 350	47	NULL	NULL	NULL
FR 450	48	NULL	NULL	NULL
FS 100	33	NULL	NULL	NULL

Intéressant non ? On voit que certains produits n'ont pas de tarif. Bon, vu la taille du jeu de données relativ aux prix de vente, il fallait s'y attendre.

Pour la suite de la démonstration, je vous propose d'éliminer les produits n'ayant pas de tarif, du coup remplacez « LEFT JOIN » par « INNER JOIN » et relancez la requête :

code_produit	produit_id	prd_produit_id	date_deb	prix
BR 500	80	80	2017-01-01	739.97000
BR 550	81	81	2017-01-01	756.69000
BR 600	82	82	2017-01-01	848.66000
BT 45 Tarière	27	27	2017-01-01	581.10000
GB 370 S	36	36	2017-01-01	1040.97000
GE 103	11	11	2017-01-01	268.39000
GE 105	26	26	2017-01-01	325.25000
GE 150	55	55	2017-01-01	394.65000
GE 250	56	56	2017-01-01	517.56000

OK, c'est mieux comme ça.

Maintenant, si vous observez le résultat obtenu, vous constatez que tous les prix de vente sont définis à la même date, soit le 1^{er} janvier 2017. Que se passerait-il si on avait un autre prix de vente au 1^{er} février 2017 ? Pour le savoir, je vous propose d'injecter artificiellement un jeu de données dans la table des prix de vente. Pour faire cela facilement, nous pouvons écrire une requête qui va réaliser les actions suivantes :

- sélectionner les prix de vente existants,
- appliquer à chaque prix de vente une augmentation de 10%
- injecter les nouveaux prix avec comme date d'application le 1^{er} février 2017

Une première requête pour préparer le terrain :

```

1 | SELECT prd_produit_id,
2 |   '2017-02-01' as new_date_deb,
3 |   prix as ancien_prix,
4 |   prix*1.1 as nouveau_prix
5 | FROM prd_prixstd
6 | WHERE date_deb = '2017-01-01';

```

Petit échantillon du résultat obtenu :

prd_produit_id	new_date_deb	ancien_prix	nouveau_prix
60	2017-02-01	459.03000	504.933000
61	2017-02-01	500.84000	550.924000
62	2017-02-01	584.45000	642.895000
51	2017-02-01	1109.00000	1219.900000
52	2017-02-01	1325.00000	1457.500000
53	2017-02-01	1620.00000	1782.000000
75	2017-02-01	2390.00000	2629.000000
76	2017-02-01	2910.00000	3201.000000
77	2017-02-01	3415.00000	3756.500000
126	2017-02-01	1019.23000	1121.153000
124	2017-02-01	868.73000	955.603000
125	2017-02-01	952.34000	1047.574000

Ok, c'est bon... Allez hop, je réutilise ma requête SELECT (en retirant quand même la colonne « ancien prix »), et j'injecte le résultat obtenu dans la table des prix de vente :

```

INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix)
SELECT prd_produit_id, '2017-02-01', prix * 1.1
FROM prd_prixstd
WHERE date_deb = '2017-01-01';

```

PHPMyAdmin a l'air plutôt content, c'est bon signe 😊

✓ 57 lignes insérées.

Identifiant de la ligne insérée : 171 (traité en 0,0000 seconde(s))

```
INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix) SELECT prd_produit_id, '2017-02-01', prix * 1.1 FROM prd_prixstd WHERE date_deb = '2017-01-01'
```

Relançons notre requête d'affichage des produits avec leurs prix :

```
SELECT a.codeproduit AS code_produit,
       a.id AS produit_id,
       b.prd_produit_id, b.date_deb, b.prix
  FROM prd_produit a
 INNER JOIN prd_prixstd b
    ON a.id = b.prd_produit_id
 ORDER BY a.codeproduit, b.date_deb
;
```

Oups ! Y'aurait pas un petit problème ?

code_produit	produit_id	prd_produit_id	date_deb	prix
BR 500	80	80	2017-01-01	739.97000
BR 500	80	80	2017-02-01	813.96700
BR 550	81	81	2017-01-01	756.69000
BR 550	81	81	2017-02-01	832.35900
BR 600	82	82	2017-01-01	848.66000
BR 600	82	82	2017-02-01	933.52600
BT 45 Tarière	27	27	2017-01-01	581.10000
BT 45 Tarière	27	27	2017-02-01	639.21000
GB 370 S	36	36	2017-01-01	1040.97000
GB 370 S	36	36	2017-02-01	1145.06700
GE 103	11	11	2017-01-01	268.39000
GE 103	11	11	2017-02-01	295.22900

Je comptais obtenir une ligne par produit, avec pour chaque produit son tarif, mais là je me retrouve avec deux lignes par produit.

J'en vois déjà qui vont me rétorquer...

« on n'a qu'à considérer que le tarif en vigueur, c'est celui du 1^{er} février. Du coup on met une clause WHERE avec sélection des tarifs au 1^{er} février, et le tour est joué !! »

C'est bien essayé, mais je ne suis pas d'accord. Certes, notre augmentation de février est une augmentation en masse, qui a été appliquée de manière uniforme sur les produits déjà tarifés du catalogue. Mais il s'agissait là d'un cas d'école. Il est rare que, dans la « vraie vie », on fasse des augmentations de ce type. Dans la « vraie vie », on applique des modifications de tarif au cas par cas, quelquefois pour une seule activité, quelquefois pour une ou plusieurs familles de produits, quelquefois produit par produit. Et tout cela à des dates très souvent différentes. C'est quoi dans ce cas, le « tarif en vigueur » ?

Pour essayer de cerner le problème, je vous propose d'appliquer quelques augmentations sur les produits ayant les identifiants 80, 81 et 82. Mais pour se rapprocher de la « vraie vie », je ne vais pas appliquer ces augmentations de manière uniforme : quelquefois j'augmenterai un seul

produit, quelquefois deux, quelquefois les 3. Je vais étaler ces augmentations du 1^{er} mars au 1^{er} juin 2017 :

```
INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix)
SELECT prd_produit_id, '2017-03-01', prix * 1.05
FROM prd_prixstd
WHERE date_deb = '2017-02-01'
and prd_produit_id in (80, 81, 82)
;

INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix)
SELECT prd_produit_id, '2017-03-15', prix * 1.02
FROM prd_prixstd
WHERE date_deb = '2017-03-01'
and prd_produit_id in (80, 82)
;

INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix)
SELECT prd_produit_id, '2017-04-01', prix * 1.02
FROM prd_prixstd
WHERE date_deb = '2017-03-15'
and prd_produit_id in (81, 82)
;

INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix)
SELECT prd_produit_id, '2017-05-01', prix * 1.02
FROM prd_prixstd
WHERE date_deb = '2017-04-01'
and prd_produit_id in (82)
;

INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix)
SELECT prd_produit_id, '2017-06-01', prix * 1.1
FROM prd_prixstd
WHERE date_deb = '2017-05-01'
and prd_produit_id in (80, 81, 82)
;

INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix)
SELECT prd_produit_id, '2017-06-01', prix * 1.1
FROM prd_prixstd
WHERE date_deb = '2017-03-01'
and prd_produit_id in (80, 81)
```

OK, relançons maintenant la requête de consultation en nous focalisant uniquement sur nos 3 produits :

```
SELECT a.codeproduit as code_produit,
       a.id as produit_id,
       b.prd_produit_id, b.date_deb, b.prix
FROM prd_produit a
INNER JOIN prd_prixstd b
    ON a.id = b.prd_produit_id
WHERE prd_produit_id in (80, 81, 82)
ORDER BY a.codeproduit, b.date_deb
;
```

Cette fois ci il ne s'agit plus d'un échantillon, mais du jeu de données complet correspondant aux 3 produits BR500, BR550 et BR600 :

code_produit	produit_id	prd_produit_id	date_deb	prix
BR 500	80	80	2017-01-01	739.97000
BR 500	80	80	2017-02-01	813.96700
BR 500	80	80	2017-03-01	854.66535
BR 500	80	80	2017-03-15	871.75866
BR 500	80	80	2017-06-01	940.13189
BR 550	81	81	2017-01-01	756.69000
BR 550	81	81	2017-02-01	832.35900
BR 550	81	81	2017-03-01	873.97695
BR 550	81	81	2017-06-01	961.37465
BR 600	82	82	2017-01-01	848.66000
BR 600	82	82	2017-02-01	933.52600
BR 600	82	82	2017-03-01	980.20230
BR 600	82	82	2017-03-15	999.80635
BR 600	82	82	2017-04-01	1019.80248
BR 600	82	82	2017-05-01	1040.19853
BR 600	82	82	2017-06-01	1144.21838

Supposons que je souhaite connaître les tarifs applicables pour ces 3 produits, en date du 20 mars 2017. Pour ce faire, j'ai besoin de ma requête initiale, mais j'ai besoin de la compléter d'une sous-requête scalaire, placée au niveau de la clause WHERE, et qui va avoir pour rôle de rechercher pour chaque produit, le tarif le plus proche de la date demandée.

Voici ce qu'il faut écrire, avec la sous-requête scalaire « stabilisée » en jaune :

```
SELECT a.codeproduit AS code_produit,
       a.id AS produit_id,
       b.prd_produit_id, b.date_deb, b.prix
  FROM prd_produit a
 INNER JOIN prd_prixstd b
    ON a.id = b.prd_produit_id
   AND b.date_deb =
      (SELECT MAX(c.date_deb) FROM prd_prixstd c
       WHERE c.prd_produit_id = b.prd_produit_id
         AND c.date_deb <= '2017-03-20')
 WHERE prd_produit_id IN (80, 81, 82)
 ORDER BY a.codeproduit, b.date_deb
;
```

Dans la sous-requête scalaire (stabilisée) de recherche du tarif le plus proche de la date d'application souhaitée (qui est ici le 20 mars 2017), vous noterez la présence de la fonction MAX(). Cette fonction est très importante, sans elle il est impossible de récupérer la date d'application la plus proche de la date souhaitée. L'autre élément important, c'est l'opérateur de comparaison « <= » (inférieur ou égal) qui permet de sélectionner la date antérieure la plus proche de celle recherchée (elle peut dans certains cas être égale à la date recherchée).

Notre requête est paramétrée pour une recherche au 20 mars 2017, alors testons-la :

code_produit	produit_id	prd_produit_id	date_deb	prix
BR 500	80	80	2017-03-15	871.75866
BR 550	81	81	2017-03-01	873.97695
BR 600	82	82	2017-03-15	999.80635

Nous récupérons deux prix en date du 15 mars, et un prix en date du 1^{er} mars (pour le produit BR550 qui n'a pas de prix au 15 mars). Cela semble bien fonctionner, alors poursuivons les tests.

Modifions la date (en rouge) dans la requête, en la paramétrant pour le 10 mars 2017 (soit avant l'augmentation du 15) ?

code_produit	produit_id	prd_produit_id	date_deb	prix
BR 500	80	80	2017-03-01	854.66535
BR 550	81	81	2017-03-01	873.97695
BR 600	82	82	2017-03-01	980.20230

Et au 10 juin 2017, ça donne quoi ?

code_produit	produit_id	prd_produit_id	date_deb	prix
BR 500	80	80	2017-06-01	940.13189
BR 550	81	81	2017-06-01	961.37465
BR 600	82	82	2017-06-01	1144.21838

Si vous prenez la peine de regarder de quelle manière certaines applications e-commerce gèrent les tarifs, vous vous apercevrez que ces applications utilisent des méthodes bien archaïques, si on les compare à ce que nous venons d'écrire. Certaines applications e-commerce ne savent tout simplement pas gérer de données tarifaires avec dates d'applications multiples, ou alors elles proposent un système tellement contraignant que les utilisateurs ne peuvent tout simplement pas s'en servir. Ces derniers se trouvent dès lors obligés de modifier les tarifs le jour J, éventuellement à minuit, pour application dès l'heure d'ouverture... Mais au fait, ça n'a pas d'heure d'ouverture un site e-commerce, c'est généralement ouvert en 7J/7 24H/24 !!

Quelquefois vous pourrez trouver des applications proposant une gestion de produits multi-dates, mais c'est le code de récupération des informations qui sera pourri. Car au lieu d'utiliser les techniques de jointure que nous venons d'étudier, les développeurs auront utilisé 2 requêtes distinctes traitées dans cet ordre :

- exécution d'une requête permettant de récupérer la liste des produits
- exécution d'une boucle PHP parcourant le résultat de la 1^{ère} requête et exécutant une nouvelle requête à chaque itération pour récupérer le tarif le plus proche

Dans ce que je viens de décrire, on peut trouver différentes manières de procéder, allant du pire au meilleur. Le pire étant le code SQL créé par concaténation sans utiliser la technique du SQL paramétré (d'un point de vue de la sécurité, c'est l'horreur).

Le seul cas qui pourrait justifier de ne pas utiliser la technique présentée dans ce chapitre, ce serait le cas où la table des produits se trouverait dans une base de données, et la table des tarifs se trouverait dans une autre base de données. Dans un cas comme celui là, impossible d'établir une jointure simple entre les tables « produit » et « prix de vente », il vaut mieux envisager l'utilisation de 2 requêtes distinctes.

Avant de poursuivre, je vous propose d'ajouter à notre requête précédente les libellés « famille » et « activité » de chaque produit :

```
SELECT a.codeproduit as code_produit,
       pf.nom as nom_famille,
       (SELECT b.nom FROM prd_activite b
        WHERE pf.prd_activite_id = b.id) as nom_activite,
       b.date_deb as date_tarif, b.prix as prix_vente
FROM prd_produit a
INNER JOIN prd_famille pf
        ON a.prd_famille_id = pf.id
INNER JOIN prd_prixstd b
        ON a.id = b.prd_produit_id
```

```

and b.date_deb = (
    SELECT MAX(c.date_deb) FROM prd_prixstd c
    WHERE c.prd_produit_id = b.prd_produit_id
    AND c.date_deb <= '2017-03-20'
)
WHERE a.prd_famille_id between 1 and 100
ORDER BY a.codeproduit, b.date_deb
;

```

Si vous avez trouvé ce chapitre ardu, vous avez bien raison. Vous avez le droit de « laisser décanter » tout ça et d'y revenir plus tard. S'il vous reste un peu de courage, vous pouvez atteler à ce qui suit.

Voyons une mise en application rapide de ce que nous venons d'étudier, transposé dans le petit monde de PHP.

Voici un exemple de ce que nous souhaitons obtenir :

Catalogue des produits avec prix de vente

Critères de sélection

Filtre / famille	Ou Filtre / Produit	Date tarif		
Broyeurs	FR 130 T	27/07/2017	▼	Valider
Code produit	Famille	Activité	Date tarif	Tarif vente
FR 130 T	Débroussailleuses à dos	Motoculture		0,00
GB 370	Broyeurs	Motoculture		0,00
GB 370 S	Broyeurs	Motoculture	2017-02-01	1 145,07
GE 103	Broyeurs	Motoculture	2017-02-01	295,23
GE 105	Broyeurs	Motoculture	2017-02-01	357,78
GE 150	Broyeurs	Motoculture	2017-02-01	434,12
GE 250	Broyeurs	Motoculture	2017-02-01	569,32
GE 35 L	Broyeurs	Motoculture		0,00

Détail de la requête SQL

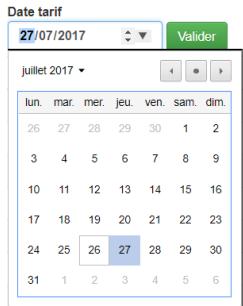
```
SELECT a.codeproduit as code_produit,
       pf.nom as nom_famille,
       (SELECT b.nom FROM prd_activite b
        WHERE pf.prd_activite_id = b.id) as nom_activite,
       b.date_deb as date_tarif, b.prix as prix_vente
  FROM prd_produit a
 INNER JOIN prd_famille pf
   ON a.prd_famille_id = pf.id
 LEFT JOIN prd_prixstd b
   ON a.id = b.prd_produit_id
  and b.date_deb =
    (SELECT MAX(c.date_deb) FROM prd_prixstd c
     WHERE c.prd_produit_id = b.prd_produit_id
     AND c.date_deb <= ?
    )
 WHERE a.prd_famille_id = ? OR a.codeproduit = ?
 ORDER BY a.codeproduit, b.date_deb;
Paramètres : 2017-07-27 | 3 | FR 130 T
```

Dans la version du script que je vous propose ici, je n'ai pas intégré de mécanisme de pagination, ceci afin de ne pas trop compliquer l'exemple, et afin que vous puissiez vous l'approprier plus facilement. Vous pourrez ajouter le mécanisme de pagination dans un second temps, quand vous vous sentirez suffisamment à l'aise avec cet exemple.

A noter que j'ai dû ajouter à la classe GenForm une nouvelle méthode, qui est la suivante :

```
public function gen_input_date($nom_champ, $valeur_champ = '',
                               $required = false, $placeholder = '', $disabled=false)
{
    return $this->gen_form_input('date', $nom_champ, $valeur_champ,
                                $required, $placeholder, $disabled);
}
```

Je rappelle que les champs de type « date » fonctionnent bien avec Chrome, mais pas avec Firefox. Vous obtiendrez donc une assistance à la saisie avec Chrome (comme celle ci-dessous), mais pas avec Firefox :



Pour les navigateurs autres que Chrome, vous devrez recourir à un plugin Javascript de type « date picker ». On en trouve de toutes sortes sur internet, mais je laisse ce sujet de côté car nous sommes dans un cours PHP.

Voici le code source du script :

```
<?php
require_once 'library/GenForm.php';
require_once 'library/HTMLPage.php';
require_once 'library/Sanitize.php';
require_once 'library/DBConnect.php';
require_once 'library/DBWrapper.php';

echo HTMLPage::getHeader('Produits et tarifs');
echo HTMLPage::getMenu();

// création d'une instance de PDO
$db = DBConnect::open();

echo '<div class="container">' . PHP_EOL;
echo '<h2>Catalogue des produits avec prix de vente</h2>', PHP_EOL;

echo '<form id="listprod" method="get">', PHP_EOL;
echo '<fieldset>', PHP_EOL;
echo '<legend>Crit&egrave;res de s&acute;lection</legend>', PHP_EOL;

// Requête sélectionnant uniquement les familles rattachées à au moins un produit
$req_familles = <<<BLOC_SQL
SELECT id, nom
FROM prd_famille
WHERE id IN (SELECT prd_famille_id FROM prd_produit)
ORDER BY nom
BLOC_SQL;
```

```

$datas_fam = DBWrapper::selectBlock($db, $req_familles);
// Préparation d'un tableau adapté au format attendu par $form->gen_input_select()
$liste_fam_for_select = [];
$liste_fam_for_select[''] = 'Sélectionnez une famille'; // 1ère option "vide"
foreach ($datas_fam as $data_fam) {
    $liste_fam_for_select[$data_fam['id']] = $data_fam['nom'];
}

$form = new GenForm();
echo '<label>Filtre / famille ' // Code famille facultatif
echo $form->gen_input_select('rech_fam', $liste_fam_for_select,
Sanitize::blinder_get('rech_fam'));
echo '</label>';
echo '&nbsp;';
echo '<label>Ou Filtre / Produit' // Code produit facultatif
echo $form->gen_input_text('rech_prod', Sanitize::blinder_get('rech_prod'), false,
    'Code Produit');
echo '</label>';
echo '&nbsp;';
echo '<label>Date tarif ' // Date de tarif obligatoire
echo $form->gen_input_date('date_tarif', Sanitize::blinder_get('date_tarif'), true);
echo '</label>';
echo '&nbsp;';
echo '<input type="submit" value="Valider" class="btn btn-success" />';
echo '</form>';

echo '<br/>', PHP_EOL;

if (count($_GET) > 0) {
    // Tableau servant à générer la clause WHERE de la requête principale
    $sql_where = [];
    // Tableau des paramètres transmis à PDO pour exécution de la requête finale
    $params = array();
    // Date de tarif en 1ère position dans la requête, donc en 1ère position
    // en 1ère position également dans le tableau des paramètres
    $params [] = Sanitize::blinder_get('date_tarif');
    // Ajout des filtres facultatifs sur famille et code produit
    $tmp_var = Sanitize::blinder_get('rech_fam');
    if ($tmp_var != '') {
        $params [] = $tmp_var;
        $sql_where []= 'a.prd_famille_id = ?';
    }
    $tmp_var = Sanitize::blinder_get('rech_prod');
    if ($tmp_var != '') {
        $params [] = $tmp_var;
        $sql_where []= 'a.codeproduit = ?';
    }
    // Les paramètres facultatifs (famille et produit) sont liés par une clause OR
    $sql_where_final = implode(' OR ', $sql_where);
    // Si aucun filtre n'est sélectionné, alors on n'ajoute pas la clause WHERE
    if ($sql_where_final != '') {
        $sql_where_final = 'WHERE ' . $sql_where_final;
    }
}

$requete = <<<BLOC_SQL
SELECT a.codeproduit as code_produit,
pf.nom as nom_famille,
(SELECT b.nom FROM prd_activite b
    WHERE pf.prd_activite_id = b.id) as nom_activite,
b.date_deb as date_tarif, b.prix as prix_vente
BLOC_SQL

```

```

FROM prd_produit a
INNER JOIN prd_famille pf
  ON a.prd_famille_id = pf.id
LEFT JOIN prd_prixstd b
  ON a.id = b.prd_produit_id
  and b.date_deb = (
    SELECT MAX(c.date_deb) FROM prd_prixstd c
    WHERE c.prd_produit_id = b.prd_produit_id
    AND c.date_deb <= ?
  )
{$sql_where_final}
ORDER BY a.codeproduit, b.date_deb
BLOC_SQL;

echo <<<BOTABLE
<table class="table table-striped table-bordered" summary = "Liste des produits" >
<thead>
<tr>
<th>Code produit</th>
<th>Famille</th>
<th>Activité</th>
<th class="text-center">Date tarif</th>
<th class="text-right">Tarif vente</th>
</tr>
</thead>
<tbody>
BOTABLE;

$datas = DBWrapper::selectBlock($db, $requete, $params);
foreach($datas as $data) {
    // Formatage du prix selon la notation française
    $data['prix_vente'] = number_format($data['prix_vente'], 2, ',', ' ');
    echo <<<TABLEROW
<tr>
<td>{$data['code_produit']}

```

4.9.4 Mini-CRUD, mais il fait le maximum

CRUD est l'acronyme de « Create Retrieve Update Delete », soit les 4 opérations de base que l'on applique à une entité « métier ». Cette entité « métier », ce peut être le catalogue des produits, la liste des clients ou des fournisseurs, la liste des tarifs vente ou achats, la liste des employés de la société, etc..

A partir du moment où il y a un CRUD, il y a du stockage et de la mise à jour de données. Et dans ce cas de figure, on trouve difficilement plus pratique qu'une base de données utilisant le langage d'interrogation SQL. Si l'on met en relation l'acronyme CRUD avec les types de requêtes SQL disponibles, nous avons :

- Pour le C de Create : la requête SQL de type INSERT
- Pour le R de Retrieve : la requête SQL de type SELECT
- Pour le U de Update : la requête SQL de type UPDATE
- Pour le D de Delete : la requête SQL de type DELETE

Voilà, ce n'est pas plus compliqué que ça.

Dans la suite de ce chapitre, nous allons construire un petit CRUD tout simple, pour découvrir les mécanismes en jeu dans ce type de module. Pour ce faire :

- nous allons créer une base MySQL et une table pour le stockage de données,
- nous allons utiliser Bootstrap et quelques une de ses classes CSS pour obtenir un rendu sympa des pages
- nous utiliserons une approche très procédurale pour cette première version, pour une approche plus orientée objet,

Nous verrons au travers de ce premier exemple que l'architecture utilisée ne va pas sans poser quelques problèmes. Elle a néanmoins le mérite de fonctionner correctement, et elle peut permettre de développer rapidement des applications fonctionnelles et robustes. Ce type d'architecture (procédurale et un peu monolithique) a beaucoup été utilisée, jusqu'au milieu des années 2000, avant d'être petit à petit remplacée par une architecture de type MVC (Modèle, Vue, Contrôleur), popularisées par des frameworks comme Ruby On Rails, Zend Framework, et quelques autres. L'architecture MVC offre certains avantages, mais aussi son lot de complications, que je ne vais pas développer pour le moment.

Nous allons créer une table des auteurs, elle constituera notre entité « métier », sur laquelle nous allons bâtir notre module CRUD. Voici le code source SQL à exécuter dans l'éditeur de code de PhpMyAdmin :

```
--  
-- Table structure for table `auteurs`  
--  
CREATE TABLE IF NOT EXISTS `auteurs` (  
    `id` int(11) NOT NULL,  
    `name` varchar(30) NOT NULL,  
    `email` varchar(30) NOT NULL,  

```

Les noms que j'ai utilisés dans le jeu d'essai ci-dessus correspondent à des personnes qui existent réellement. Ce sont tous des auteurs d'ouvrages qui m'ont apporté quelque chose, et pour lesquels je me sens redevable. J'avais envie de leur faire ce petit clin d'œil. A noter que les adresses email sont « bidons », mais que le nom de domaine constitue un indice du domaine dans lequel ces auteurs ont écrit.

Tous les éléments étant en place, voici la première version du script « index.php » :

```
<?php
require 'library/DBConnect.php';
require 'library/HTMLPage.php';
echo HTMLPage::getHeader('CRUD Exemple 1');
$pdo = DBConnect::open();
?>
<div class="container">
    <div class="row">
        <h3>PHP CRUD Grid</h3>
    </div>
    <div class="row">
        <table class="table table-striped table-bordered">
            <thead>
                <tr>
                    <th>Nom</th>
                    <th>Email</th>
                    <th>Téléphone</th>
                </tr>
            </thead>
            <tbody>
                <?php
                    $sql = 'SELECT * FROM auteurs ORDER BY name ASC';
                    foreach ($pdo->query($sql) as $row) {
                        echo '<tr>' . PHP_EOL;
                        echo '<td>' . $row['name'] . '</td>' . PHP_EOL;
                        echo '<td>' . $row['email'] . '</td>' . PHP_EOL;
                        echo '<td>' . $row['mobile'] . '</td>' . PHP_EOL;
                        echo '</tr>' . PHP_EOL;
                    }
                ?>
            </tbody>
        </table>
    </div>
</div> <!-- /container -->
<?php
DBConnect::close();
echo HTMLPage::getFooter();
```

Voici le résultat obtenu pour cette première version :

PHP CRUD Grid

Nom	Email	Téléphone
Czes Kosniowski	czes@c64.com	0101010101
Daniel Shiffman	dsh@p5iscool.com	0808020202
Ilya Virgatchik	ilv@basic.com	0402020202
Jon Millington	jmill@curve.com	0606060606
Kyle Simpson	kyl@ydkjs.com	0302030202
Robert Dosny	rdon@graphisme.fr	0502020202
Roger LeMasne	rlm@polyedre.fr	0202020202

PHP - Premiers pas || CRUD Tutorial

Ca se présente plutôt bien, il est temps de placer les boutons permettant de déclencher les 4 opérations de notre module CRUD. Voici le type de rendu que j'aimerais obtenir.

Gestion des auteurs

Gestion des auteurs			
Nom	Email	Téléphone	Action
Czes Kosniowski	czes@c64.com	0101010101	<button>Afficher</button> <button>Modifier</button> <button>Supprimer</button>
Daniel Shiffman	dsh@p5iscool.com	0808020202	<button>Afficher</button> <button>Modifier</button> <button>Supprimer</button>
Ilya Virgatchik	ilv@basic.com	0402020202	<button>Afficher</button> <button>Modifier</button> <button>Supprimer</button>
Jon Millington	jmill@curve.com	0606060606	<button>Afficher</button> <button>Modifier</button> <button>Supprimer</button>
Kyle Simpson	kyl@ydkjs.com	0302030202	<button>Afficher</button> <button>Modifier</button> <button>Supprimer</button>
Robert Dosny	rdon@graphisme.fr	0502020202	<button>Afficher</button> <button>Modifier</button> <button>Supprimer</button>
Roger LeMasne	rlm@polyedre.fr	0202020202	<button>Afficher</button> <button>Modifier</button> <button>Supprimer</button>

Pour le bouton de création, je vous propose de placer ce code juste au dessus de la balise « table » :

```
<p>
  <a href="auteur_crud.php?action=c" class="btn btn-success">Créer</a>
</p>
```

Nous allons également ajouter une colonne « Action » dans l'entête du tableau HTML :

```
<thead>
<tr>
  <th>Nom</th>
  <th>Email</th>
  <th>Téléphone</th>
  <th>Action</th>
</tr>
</thead>
```

Et pour finir, nous allons ajouter une nouvelle colonne dans la partie « tbody » du tableau HTML, juste avant la balise « /tbody ».

```
echo '<td class="text-center">';
echo '<a class="btn btn-sm btn-success"
href="auteur_crud.php?action=r&id='.$row['id']. '">Afficher</a>&ampnbsp';
echo '<a class="btn btn-sm btn-warning"
href="auteur_crud.php?action=u&id='.$row['id']. '">Modifier</a>&ampnbsp';
echo '<a class="btn btn-sm btn-danger"
```

```
href="auteur_crud.php?action=d&id='".$row['id']."'>Supprimer</a>';
echo '</td>';
```

Vous aurez probablement remarqué que toutes les balises « a » de la nouvelle colonne utilisent un attribut « href » qui a pratiquement toujours le même « motif » :

- toutes les balises font appel au même script « auteur_crud.php » qui reçoit 2 paramètres
- le premier paramètre s'appelle « action » et il reçoit l'un des codes « c », « r », « u » ou « d »
- le seconde paramètre s'appelle « id » et il est alimenté avec l'identifiant de chaque auteur, sauf dans le cas de l'action « c », qui correspond à la création d'un nouvel auteur

Je souhaite donc que le même script « auteur_crud.php » soit en mesure de gérer la création, la modification, l'affichage et la suppression. Voici quelques copies d'écran pour vous aider à comprendre l'idée.

- en mode affichage, je souhaite obtenir l'affichage ci-dessous (les champs sont grisés car verrouillés, l'utilisateur ne peut que consulter et cliquer sur « retour »)

Affichage d'un auteur

Name	Czes Kosniowski
Email Address	czes@c64.com
Mobile Number	0101010101
Retour	

- en mode création :

Création d'un auteur

Name	Saisissez un nom
Email Address	Saisissez une adresse mail
Mobile Number	Saisissez un mobile
Valider la Création Annulation	

- en mode modification :

Modification d'un auteur

Name
Czes Kosniowski

Email Address
czes@c64.com

Mobile Number
0101010101

Valider la Modification **Annulation**

- en mode suppression (tous les champs de saisie sont verrouillés en consultation) :

Suppression d'un auteur

Name
Czes Kosniowski

Email Address
czes@c64.com

Mobile Number
0101010101

Valider la Suppression **Annulation**

En mode modification comme en mode création, la saisie de l'utilisateur doit être contrôlée :

Création d'un auteur

Name
Saisissez un nom

Zone obligatoire

Email Address
xxxx

Adresse email incorrecte

Mobile Number
Saisissez un mobile

Zone obligatoire

Valider la Création **Annulation**

Voici le code source du script « index.php » :

```

<?php
require 'library/DBConnect.php';
require 'library/HTMLPage.php';
echo HTMLPage::getHeader('CRUD Exemple 1');
$pdo = DBConnect::open();
?>


### Gestion des auteurs



Créer



| Nom | Email | Téléphone | Action |
|-----|-------|-----------|--------|
|-----|-------|-----------|--------|


```

Nous allons maintenant nous pencher sur le script « auteur_crud.php ».

Attention, c'est un gros morceau, du fait qu'il fait plusieurs choses. Il gère en effet :

- le premier affichage du formulaire (aussi bien en création, qu'en modification, suppression ou affichage)
- le contrôle de validité dans le cas de modification et création
- le verrouillage des champs de saisie dans le cas d'affichage et de suppression (avec en prime le masquage du bouton « valider » dans le cas de l'affichage)
- le processus lié à la mise à jour (en modification, création et suppression) est le point le plus délicat à comprendre, et donc à gérer.

En effet, il convient de savoir à coup sûr ...

- si on se trouve dans le premier affichage : c'est le cas quand on arrive du script « index.php » via une requête HTTP de type GET, et dans ce cas de figure, on se contente de lire la base de données, d'afficher le formulaire, et d'attendre une action de l'utilisateur
- si on se trouve en phase de validation de la saisie, c'est-à-dire si l'utilisateur a cliqué sur le bouton « valider » du formulaire. Il y a plusieurs manières de s'en assurer, le moyen que j'ai retenu consiste à placer le code action (« c », « r », « u » ou « d ») en zone cachée du formulaire. Lors du premier affichage du formulaire, le code action existe dans \$_GET mais pas dans \$_POST. Lors de la phase de validation du formulaire, le code action existe dans \$_POST.
 - o Si dans la phase de validation du formulaire, aucune anomalie n'est constatée dans la saisie de l'utilisateur, alors on met à jour la base de données et on se redirige vers « index.php »
 - o Si au contraire des anomalies sont constatées, on réaffiche le formulaire avec ses messages d'erreur, et on attend une nouvelle action de la part de l'utilisateur

Je n'ai probablement pas mis autant de commentaires qu'il aurait fallu pour un code comme celui-ci. En réalité, malgré sa longueur, il n'est pas si compliqué que cela, mais il y a quelques petites difficultés que vous parviendrez à éliminer en testant le code.

Vous noterez que, pour la génération des champs de saisie du formulaire, j'ai utilisé la classe GenForm que nous avons étudiée au chapitre 4.4.5.2. Elle est très pratique, c'eut été dommage de ne pas s'en servir ici 😊.

Voici le code source du script « auteur_crud.php ».

```

<?php
require 'library/DBConnect.php';
require 'library/HTMLPage.php';
require 'library/GenForm.php';

if (isset($_GET['action'])) {
    $action = trim(strip_tags($_GET['action']));
    if (!in_array($action, array('c', 'r', 'u', 'd'))) {
        // envoi d'un message dans la log de PHP
        error_log('Code action CRUD incorrect, redirection sur 404.html');
        header('404.html');
        exit();
    }
    if ($action != 'c' && !isset($_GET['id'])) {
        // envoi d'un message dans la log de PHP
        error_log('Code id CRUD manquant, redirection sur 404.html');
        header('404.html');
        exit();
    }
    $id = 0;
    if (in_array($action, array('r', 'u', 'd'))) {
        $id = (int)$_GET['id'];
    }
} else {
    // envoi d'un message dans la log de PHP
    error_log('Code action CRUD manquant, redirection sur 404.html');
    header('404.html');
    exit();
}

$liste_titres = array('c'=>'Création', 'r'=>'Affichage', 'u'=>'Modification',
                      'd'=>'Suppression');
$titre = $liste_titres[$action];
if ($action == 'd' || $action == 'r') {
    $disabled = true;
} else {
    $disabled = false;
}

$pdo = DBConnect::open();

$errors = [];
$fields = [];
$fieldskey = array('name', 'email', 'mobile');
$valid = true;

foreach ($fieldskey as $fieldkey) {
    $fields[$fieldkey] = '';
}
if ($id != 0) {
    $sql = "SELECT name, email, mobile FROM auteurs WHERE id = ?";
    $st = $pdo->prepare($sql);
    $st->execute(array($id));
    if ($st) {
        $fields = $st->fetch(PDO::FETCH_ASSOC);
    }
}

```

```

// si la méthode HTTP est de type POST, alors on est dans la phase de validation
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

    // on ne fait les contrôles de saisie qu'en modification et création
    if ($_POST['action'] == 'u' || $_POST['action'] == 'c') {
        foreach ($fieldskey as $fieldkey) {
            $fields[$fieldkey] = isset($_POST[$fieldkey]) ?
                trim(strip_tags($_POST[$fieldkey])) : '';
        }
    }

    foreach ($fields as $fieldkey => $fieldval) {
        if ($fieldval == '') {
            $errors[$fieldkey] = 'Zone obligatoire';
            $valid = false;
        } else {
            if ($fieldkey == 'email') {
                if (!filter_var($fieldval, FILTER_VALIDATE_EMAIL)) {
                    $errors[$fieldkey] = 'Adresse email incorrecte';
                    $valid = false;
                }
            }
        }
    }
}

if ($valid) {
    if ($id != 0) {
        if ($action == 'd') {
            $sql = "DELETE FROM auteurs WHERE id = ?";
            $st = $pdo->prepare($sql);
            $st->execute(array($id));
        }
        if ($action == 'u') {
            $sql = "UPDATE auteurs SET name=?, email=?, mobile=? WHERE id = ?";
            $st = $pdo->prepare($sql);
            $st->execute(array($fields['name'], $fields['email'],
                $fields['mobile'], $id));
        }
    } else {
        // $action == 'c'
        $sql = "INSERT INTO auteurs (name, email, mobile) values(?, ?, ?)";
        $st = $pdo->prepare($sql);
        $st->execute(array($fields['name'], $fields['email'], $fields['mobile']));
    }
}

DBConnect::close();
header("Location: auteur_list.php");
}

echo HTMLPage::getHeader('CRUD Exemple - '.$titre);
$pdo = DBConnect::open();
$inputs = new GenForm();
?>

```

```

<div class="container">
    <div class="row">
        <h3><?php echo $titre; ?> d'un auteur</h3>
    </div>
    <div class="row">
        <form class="form-horizontal" method="post">
            <?php
                // Création des champs cachés "action" et "id"

                echo $inputs->gen_input_hidden('action', $action);
                echo $inputs->gen_input_hidden('id', $id);
            ?>
            <div class="control-group">
                <label class="control-label">Name</label>
                <div class="controls">
                    <?php echo $inputs->gen_input_text('name', $fields['name'],
                        false, 'Saisissez un nom', $disabled); ?>
                    <?php if (isset($errors['name'])): ?>
                        <span class="text-danger "><?php echo $errors['name'];
                    ?></span>
                    <?php endif; ?>
                </div>
            </div>
            <div class="control-group">
                <label class="control-label">Email Address</label>
                <div class="controls">
                    <?php echo $inputs->gen_input_text('email', $fields['email'],
                        false, 'Saisissez une adresse mail', $disabled); ?>
                    <?php if (isset($errors['email'])): ?>
                        <span class="text-danger "><?php echo $errors['email'];
                    ?></span>
                    <?php endif; ?>
                </div>
            </div>
            <div class="control-group">
                <label class="control-label">Mobile Number</label>
                <div class="controls">
                    <?php echo $inputs->gen_input_text('mobile', $fields['mobile'],
                        false, 'Saisissez un mobile', $disabled); ?>
                    <?php if (isset($errors['mobile'])): ?>
                        <span class="text-danger "><?php echo $errors['mobile'];
                    ?></span>
                    <?php endif; ?>
                </div>
            </div>
            <div class="control-group">
                <br>
                <?php
                    if ($action != 'r') {
                        echo '<button type="submit" class="btn btn-warning">' .
                            'Valider la '.$titre.'</button>&ampnbsp';
                        echo '<a class="btn btn-success" href="auteur_list.php">' .
                            'Annulation</a>';
                    } else {
                        echo '<a class="btn btn-success" href="auteur_list.php">' .
                            'Retour</a>';
                    }
                ?>
            </div>

```

```
</form>
</div>

</div> <!-- /container -->
<?php
DBConnect::close();
echo HTMLPage::getFooter();
```

Ce code particulièrement dense est largement améliorable, de nombreux éléments peuvent être refactorisés de manière à être plus facilement réutilisables pour le développement d'autres modules de type CRUD. C'est un travail de fond que je vous invite à faire, une fois que vous avez compris le fonctionnement de ce script et que vous aurez fait le tour de ses principales difficultés.

J'ai eu la chance de pouvoir étudier quelques modèles de CRUD proposés dans des tutos disponibles sur le web. Le CRUD que je vous ai proposé peut sembler plus complexe que certains exemples que vous pouvez trouver sur internet. Cela est peut être dû au fait que j'ai utilisé le même script – et surtout le même formulaire – pour l'affichage, la création, la modification et la suppression (avec quelques adaptations minimes selon le contexte, comme par exemple le verrouillage des champs en mode affichage et suppression). Certains exemples que j'ai trouvés sur internet proposaient une approche différente consistant à appeler un script différent pour chaque type d'action (un pour la création, un pour la modification, etc...). Chaque script étant spécialisé dans une tâche, la solution pouvait sembler plus simple que la mienne, mais elle se traduisait en général par beaucoup de code dupliqué, et dans certains cas par des redirections alambiquées rendant le fonctionnement du module difficile à comprendre.

Il existe en réalité de nombreuses manières de développer un CRUD. Dans le chapitre suivant, je vous proposerai quelques conseils de lecture, vous permettant d'étudier de nouvelles approches.

4.9.5 Pistes pour étudier l'architecture MVC

Le module CRUD que nous avons étudié au chapitre précédent a pour avantage d'être efficace, et relativement facile à maîtriser. Mais il souffre quand même de quelques défauts, notamment en termes :

- de modularité (avec son approche quelque peu monolithique),
- de maintenabilité (pour l'intégration d'évolutions futures),
- de réutilisabilité (c'est lié à la modularité), certaines parties du code nécessiteraient d'être refactorisées et externalisées dans une librairie à part
- de testabilité, car l'approche étant monolithique, certains composants fortement couplés compliquent un peu la mise en place de tests

Pour améliorer l'architecture de notre CRUD du chapitre précédent, nous pouvons faire appel à un framework utilisant une architecture de type MVC (Modèle Vue Contrôleur). C'est ce que je vous propose d'étudier avec un cours spécifique dédié à l'étude du micro-framework SILEX (basé sur Symfony). Vous trouverez ce support de cours en téléchargement libre dans ce dépôt : https://github.com/gregja/Cours_SILEX_4_Biz

Vous trouverez également dans ce dépôt le lien vers un autre dépôt contenant le code source de l'application basée sur SILEX.

Même si SILEX est un micro-framework, il est basé sur SYMFONY qui est gros framework. Peut être recherchez-vous une solution plus légère, si c'est le cas vous trouverez sûrement votre bonheur dans ce livre :

CouchDB and PHP Web Development Beginner's Guide, de Tim Juravich,
aux éditions Packt Publishing (juin 2012)

<https://www.packtpub.com/web-development/couchdb-and-php-web-development-beginners-guide>

Dans son livre, Tim Juravich propose une classe qu'il a appelée Bones (cf. chapitre 4 du livre), classe qui permet de mettre en œuvre une architecture MVC simplifiée mais néanmoins pleinement opérationnelle. Dans son livre, Tim explique comment écrire cette classe, et bien sûr comment l'utiliser.

Tim Juravich a créé également un dépôt Github dans lequel on retrouve le code source de la classe Bones :

<https://github.com/timjuravich/bones/>

4.9.6 Manuel du petit Scraper

Nous avons déjà évoqué la notion de « Web Scraping » dans le support de cours suivant : « Javascript – Premier pas »

Pour rappel, voici un extrait de la définition Wikipédia, pour le terme « Web Scraping » :

« Le web scraping (parfois appelé harvesting) est une technique d'extraction du contenu de sites Web, via un script ou un programme, dans le but de le transformer pour permettre son utilisation dans un autre contexte, par exemple le référencement. /.../ Cela permet de récupérer le contenu d'une page web en vue d'en réutiliser le contenu. »

Je vous invite à lire la suite de l'article sur Wikipédia :

https://fr.wikipedia.org/wiki/Web_scraping

Toute personne qui a besoin de collecter des données pour mettre en lumière des faits, et pour rapprocher des données de sources différentes, peut être intéressée par le « web scraping ».

Dans les startups, le « web scraping » est souvent une activité dévolue au « growth hacker » (traduisez : « celui ou celle qui développe la croissance »).

Pour les non développeurs, il existe une kyrielle d'outils gratuits - et quelquefois payants - pour scraper les données des copains (et des « pas copains » aussi). Le plugin pour navigateur « Dataminer » est un plugin spécialisé dans le scraping de page, mais les possibilités de ce genre d'outil sont limitées, et il est souvent préférable de développer ses propres scripts.

Il existe un business du web scraping, avec des sociétés qui se sont spécialisées dans ce type d'activité, et « scrapent » des données pour d'autres sociétés.

PHP est plutôt bien outillé pour faire du « web scraping », on peut l'utiliser pour scraper une page, ou la totalité d'un site. Mais il n'est pas le seul, d'autres langages le permettent également, tels que Javascript (avec NodeJS), Python, Ruby, Perl, etc...

Bien souvent on ne fait pas étalage de ce genre d'activité. C'est en effet un secteur qui évolue en « zone grise », souvent à la limite de la légalité, d'autant que certains sites internet annoncent très clairement que le contenu de leurs pages n'est pas destiné à être scrapé, et qu'ils se réservent le droit d'exercer des poursuites contre les scrapers. D'autres se contentent de vous repérer via votre adresse IP, et de vous radier simplement de leur base d'utilisateurs, c'est par exemple le cas d'un réseau social très connu. Il convient donc de faire attention à ce que l'on fait, et de bien se renseigner sur ce qu'autorise le site dont on souhaite « scraper » des pages.

A noter qu'on trouve aussi des tutos expliquant comment se prémunir contre le scraping. J'ai parcouru celui-ci en diagonale, il m'a semblé suffisamment intéressant pour que je vous le propose (si c'est nul, vous avez le droit de m'engueuler :D) :

<https://github.com/JonasCz/How-To-Prevent-Scraping>

On m'a demandé récemment de faire du scraping de sites internet de type annuaires, en vue de constituer une base de prospects (pour de l'emailing et du phoning). Cela vous arrivera peut être aussi. Ce n'est pas une tâche facile à réaliser, car sur certains sites « annuaires », si les adresses sont relativement faciles à scraper, en revanche les numéros de téléphone sont bien souvent masqués, et ne peuvent être affichés qu'en déclenchant un peu de code Javascript, ce qu'un scraper PHP n'est pas en mesure de faire. La limite d'un scraper PHP se situe justement là, sur les données qui ne sont pas directement accessibles via le code HTML de base d'une page, et qui nécessitent le déclenchement de code JS complémentaire.

Nous allons écrire un petit scraper PHP, ce sera l'occasion de découvrir deux supers outils : cURL et XPath.

cURL, c'est un formidable logiciel, indépendant de PHP, mais qui est livré avec PHP, packagé sous la forme d'une extension, comme on peut le vérifier avec la fonction `phpinfo()` :

curl	
cURL support	enabled
cURL Information	7.54.0
Age	3
Features	
AsynchDNS	Yes
CharConv	No
Debug	No
GSS-Negotiate	No
IDN	Yes
IPv6	Yes
Krb4	No
Largefile	Yes
libz	Yes

Pour info, voici un extrait de la documentation Wikipédia :

cURL (abréviation de client URL request library : « bibliothèque de requêtes aux URL pour les clients ») est une interface en ligne de commande, destinée à récupérer le contenu d'une ressource accessible par un réseau informatique.

Source : <https://fr.wikipedia.org/wiki/CURL>

PHP fournit une documentation très complète sur l'utilisation de l'extension cURL :

<http://php.net/manual/fr/book.curl.php>

Que peut-on faire avec cURL ? Plein de choses :

- charger des pages web pour les scrapper
- lancer des requêtes HTTP de type GET, POST, PUT et DELETE, ce qui est parfait quand on veut faire appel à des [API](#) (Application Programming Interface), et « consommer » des services webs basés sur l'architecture [REST](#) (representational state transfer).

Dans notre cas, nous allons utiliser cURL pour lancer des requêtes HTTP de type GET, afin de récupérer les pages que nous souhaitons scrapper. Comme une session cURL nécessite la déclaration de plusieurs paramètres, nous allons l'encapsuler dans une fonction plus facilement réutilisable :

```
/**
 * Fonction d'exécution de requêtes HTTP de type GET
 * utilisant cURL
 * @param $url
 * @return mixed
 */
function curlGet($url) {
    $ch = curl_init(); // initialisation de la session cURL
    // Setting cURL options
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
    curl_setopt($ch, CURLOPT_USERAGENT, WEBBOT_NAME); // Webbot name
    curl_setopt($ch, CURLOPT_FOLLOWLOCATION, TRUE);
    curl_setopt($ch, CURLOPT_URL, $url);
    $results = curl_exec($ch); // Exécution de la session cURL
    curl_close($ch); // Arret de la session cURL
    return $results; // Envoi de la sortie de la session cURL
}
```

Vous noterez que la fonction fait appel à une constante globale (WEBBOT_NAME, que j'ai « stabilotée »). Comme cURL va simuler l'équivalent d'une requête HTTP telle que l'aurait lancée un vrai navigateur, nous allons demander à cURL d'utiliser le contenu de cette constante, histoire de donner du grain à moudre aux sites que nous allons scrapper (c'est surtout pour ne pas les alerter trop vite sur le fait qu'ils vont se faire scrapper). Voici le contenu de la constante que je vous propose d'utiliser :

```
define("WEBBOT_NAME", "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Mobile
Safari/537.36");
```

Pour jouer avec cURL, nous avons besoin d'une page à scrapper, je vous propose de scrapper le tableau des pays qui se trouve sur cette page Wikipédia :

https://fr.wikipedia.org/wiki/ISO_3166-1#Table_de_codage

Il y a plusieurs tableaux HTML sur cette page, celui qui m'intéresse c'est le second, qui ressemble à ça :

Num.	alpha-3	alpha-2	Codes ISO régions	Nom français	Nom ISO du pays ou territoire	Nom dans la langue originale, en forme longue (traduction en français)
004	AFG	AF	ISO 3166-2:AF	.af Afghanistan	AFGHANISTAN	د افغانستان اسلامي دوستانه اسلامی (Da Afghanestān Islami Jomhouriyyet) ; جمهوری اسلامی افغانستان (Jomhūri-ye Eslāmi-ye Afgānestān) ; (République islamique d'Afghanistan)
710	ZAF	ZA	ISO 3166-2:ZA	.za Afrique du Sud	AFRIQUE DU SUD	Republic of South Africa ; Republiek van Suid-Afrika ; IRiphablikī yaseMzantsi Afrika ; IRiphablikī yaseNingizimu Afrika ; IRiphablikī yeSewula Afrika ; (République d'Afrique du Sud)
248	ALA	AX	ISO 3166-2:AX	.ax Îles Åland	ÅLAND, ÎLES	Landskapet Åland ; Ahvenanmaan maakunta ; (État libre associé d'Åland)
...	ISO

J'aimerais extraire de ce tableau le code « alpha-3 » (en 2^{ème} colonne), et le nom du pays (celui en majuscule, dans la 6^{ème} colonne). Pour voir comment nous allons scraper ce tableau, nous devons nous intéresser au code source du tableau. Attention, j'ai bien dit le « code source », et non pas sa représentation dans le DOM. En effet, ce que cURL va récupérer, c'est le code HTML d'origine de la page, et pas sa représentation dans le DOM, telle que nous la voyons via les outils de développement du navigateur. Donc, pour être sûr de bien comprendre le code que cURL va récupérer, il est préférable de se focaliser sur le code source initial de la page. Jetons un coup d'œil à la ligne correspondant à l'Afghanistan. J'y ai « stabilisé » les colonnes que je souhaite récupérer :

```
<tr>
<td><tt>004</tt></td>
<td><tt>AFG</tt></td>
<td id="AF"><tt>AF</tt></td>
<td><small><a href="/wiki/ISO_3166-2:AF" title="ISO 3166-2:AF">ISO 3166-2:AF</a></small></td>
<td><span class="flagicon" data-sort-value="Afghanistan"><img alt="Flag of Afghanistan" href="/wiki/Fichier:Flag_of_Afghanistan.svg" class="image" title="Drapeau de l'Afghanistan"></a>&#160;</span><a href="/wiki/Afghanistan" title="Afghanistan">Afghanistan</a></span></td>
<td>AFGHANISTAN</td>
<td><span class="lang-ps" lang="ps" dir="rtl"> د افغانستان اسلامي دوستانه اسلامي </span> ( <i><span class="transcription lang-ps" lang="ps-Latin" dir="ltr">Da Afghanestān Islami Jomhouriyyet</span></i> ) &#160; ; <span class="lang-fa" lang="fa" dir="rtl"> جمهوری اسلامی افغانستان ( <i><span class="transcription lang-fa" lang="fa-Latin" dir="ltr">Jomhūri-ye Eslāmi-ye Afgānestān</span></i> ) &#160; ; (République islamique d'Afghanistan)</td>
</tr>
```

cURL est très bien pour récupérer le HTML « brut », mais pour le scraper, nous avons besoin de techniques de recherche équivalentes à celle que nous utilisons en Javascript, dans le navigateur (cf. cours « Javascript – Premiers pas »). Heureusement, PHP est fourni avec les classes DomDocument et DOMXPath. Grâce à ces deux extensions, nous bénéficions d'un jeu d'objets et de méthodes proches de celles dont nous disposons en Javascript.

Le code suivant présente un exemple de scraping adapté à la structure de la page des pays de Wikipédia. J'ai placé des commentaires pratiquement à chaque ligne, afin de vous guider le mieux possible. Pour gagner de la place, je ne remets pas le code source de la fonction curlGet, que nous avons déjà vue, voici par contre la suite du script :

```
$url = 'https://fr.wikipedia.org/wiki/ISO_3166-1#Table_de_codage';
$page = curlGet($url); // Chargement du fichier HTML via cURL

$dom = new DomDocument(); // Instanciation d'un objet DomDocument
@$dom->loadHTML($page); // Chargement du HTML dans l'objet DomDocument

// c'est le second tableau de la page qui m'intéresse
$table = $dom->getElementsByName('table')->item(1);

$xpath = new DOMXPath($dom); // Instanciation d'un objet DOM XPath
$query = 'tr'; // la requête XPath consiste à chercher les balises "tr"
$entries = $xpath->query($query, $table);

echo '<ul>';
foreach ($entries as $str_entry) {
    $tds = $str_entry->getElementsByTagName('td'); // sélection des "td"
    // test pour prévenir le cas de la ligne entête (avec balises "th")
    if (is_object($tds[1]) && is_object($tds[5])) {
        echo '<li>';
        echo $tds[1]->nodeValue . ' => ';
        echo $tds[5]->nodeValue;
        echo '</li>';
    }
}
echo '</ul>';
```

Voici un échantillon de ce que j'obtiens quand j'exécute ce code :

- AFG => AFGHANISTAN
- ZAF => AFRIQUE DU SUD
- ALA => ÅLAND, ÎLES
- ALB => ALBANIE
- DZA => ALGÉRIE
- DEU => ALLEMAGNE
- AND => ANDORRE
- AGO => ANGOLA
- AIA => ANGUILLA
- ATA => ANTARCTIQUE

Vous voyez que l'on retrouve dans le code des fonctions bien connues des développeurs JS, comme getElementsByTagName.

Pour affiner notre sélection et récupérer les balises « tr » du tableau, nous utilisons une requête XPath.

Mais au fait c'est quoi XPath ? Voici une définition que j'ai empruntée à Wikipédia :

XPath est un langage (non XML) pour localiser une portion d'un document XML. Initialement créé pour fournir une syntaxe et une sémantique aux fonctions communes à XPointer et XSL, **XPath** a rapidement été adopté par les développeurs comme langage d'interrogation simple d'emploi.

On peut ajouter à cette définition le fait que XPath est un langage normalisé par le W3C, au même titre que HTML et XML. Même si HTML est plus souple que XML, les deux normes sont en réalité très proches et XPath se révèle très pratique pour effectuer des requêtes d'interrogation dans des pages HTML.

On trouve sur internet de nombreux tutoriaux pour s'initier à XPath. On notera que le site internet W3Schools propose des tutoriaux intéressants sur le sujet :

https://www.w3schools.com/xml/xpath_intro.asp

Voici un tableau présentant les principaux éléments de la syntaxe XPath, que l'on utilise pour effectuer des requêtes sur des documents HTML et XML :

Expression	Description
/	Renvoie le noeud "racine"
//	Renvoie l'ensemble des noeuds
.	Renvoie le noeud courant
..	Renvoie le noeud parent
@	Renvoie les attributs
[n]	Renvoie le Nème élément
[last()]	Renvoie le dernier élément
[last()-n]	Renvoie le Nème noeud à partir de la fin
[position()<n]	Renvoie les N premiers éléments
[x>n]	Renvoie les X éléments contenant un élément supérieur à N

La requête XPath de notre exemple est très simple, puisque nous nous contentons de rechercher les balises « tr ». Mais on peut écrire des requêtes plus sophistiquées.

Voici quelques exemples de filtres XPath que j'avais utilisés dans un autre projet de scraping :

Requête XPath	Objectif visé
.//a[contains(@class, 'links')]	trouver tous les nœuds associés à la classe « links »
.//div[contains(@class, 'adresse')]/a	trouver les balises « a » dont la classe mère contient « adresse »

<code>.//span[contains(@class, 'num')]</code>	Trouver les balises « span » contenant une classe « num »
<code>.//li[contains(@class, 'site')]/a/@href</code>	Trouver les balises « a » associées aux éléments dont la classe contient « site », et renvoie les attributs « href » des balises « a » (l'objectif ici était de récupérer les URL des sites internets référencés sur la page).

Une alternative au code précédent aurait consisté à écrire ceci :

```
// Variante utilisant la propriété "childNodes"
echo '<ul>';
foreach ($entries as $str_entry) {
    // avec childNodes, on récupère des noeuds autres que "td",
    // du coup identifier les bons numéros de noeud prend plus de temps
    $tds = $str_entry->childNodes;

    /*
     * pour identifier les n° de noeud qui nous intéressent, on peut s'aider
     * provisoirement de la boucle suivante :
    foreach($tds as $td=>$std) {
        if ($std->nodeValue == 'AFG' || $std->nodeValue == 'AFGHANISTAN') {
            echo $tdi.' => '.$std->nodeValue;
        }
    }
    */

    // pour prévenir le cas de la ligne entête (avec balises "th")
    if (is_object($tds[2]) && is_object($tds[10])) {
        echo '<li>';
        echo $tds[2]->nodeValue . ' => ';
        echo $tds[10]->nodeValue;
        echo '</li>';
    }
}
echo '</ul>';
```

La propriété `childNodes` nous renvoie tous les noeuds contenus dans la balise « tr », aussi bien les « enfants » directs, que les noeuds plus éloignés. Du coup il faut déterminer à quel numéro de noeud se trouvent les informations qui nous intéressent. Et comme certaines balises « td » contiennent plusieurs balises « span », la détermination des bons numéros de noeuds est plus laborieuse. Pour m'aider dans cette détermination, j'avais utilisé le subterfuge suivant :

```
/*
 * pour identifier les n° de noeud qui nous intéressent, on peut s'aider
 * provisoirement de la boucle suivante :
foreach($tds as $td=>$std) {
    if ($std->nodeValue == 'AFG' || $std->nodeValue == 'AFGHANISTAN') {
        echo $tdi.' => '.$std->nodeValue;
    }
}
*/
```

Vous pourrez vous amuser à décommenter ces quelques lignes, pour voir ce que ça donne. Eh oui, quelquefois on est obligé de feinter, quand l'information se révèle difficile à trouver.

Si vous avez envie d'approfondir ce sujet du « web scraping », je vous propose quelques pistes de lecture :

Webbots, Spiders, and Screen Scrapers, 2nd Edition
A Guide to Developing Internet Agents with PHP/CURL
de Michael Schrenk
NoStarch, March 2012
<https://www.nostarch.com/webbots2>

Instant PHP Web Scraping
de Jacob Ward
Publisher: Packt Publishing (July 2013)
<https://www.packtpub.com/web-development/instant-php-web-scraping-instant>

Scraping for Journalists
de Paul Bradshaw
LeanPub, published on 2017-02-03
<http://leanpub.com/scrapingforjournalists>

Les livres de Michael Schrenk et Jacob Ward sont très complémentaires. Ils présentent des techniques différentes permettant d'arriver au même résultat. J'ai découvert beaucoup de techniques intéressantes grâce au livre de Michael Schrenk, mais les techniques de scraping que nous avons utilisées dans ce chapitre sont plus proches de celles présentées par Jacob Ward. Le livre de Paul Bradshaw est destiné à des non développeurs, mais il présente des techniques intéressantes qui peuvent inspirer les développeurs.

A signaler également ce livre, que j'ai découvert récemment et que je n'ai pas lu (mais qui semble intéressant) :

Web Scraping for PHP developers, a practical guide
de Sameer Borate
LeanPub, published on 2014-12-19
<https://leanpub.com/web-scraping>

5 Annexe

5.1 Opérateurs

PHP fournit un grand nombre d'opérateurs, qui sont décrits sur la page suivante :
<http://fr.php.net/manual/fr/language.operators.php>

Extrait de la page ci-dessus :

Sommaire

- [La priorité des opérateurs](#)
- [Les opérateurs arithmétiques](#)
- [Les opérateurs d'affectation](#)
- [Opérateurs sur les bits](#)
- [Opérateurs de comparaison](#)
- [Opérateur de contrôle d'erreur](#)
- [Opérateur d'exécution](#)
- [Opérateurs d'incrémentation et décrémentation](#)
- [Les opérateurs logiques](#)
- [Opérateurs de chaînes](#)
- [Opérateurs de tableaux](#)
- [Opérateurs de types](#)

Un opérateur est quelque chose qui prend une ou plusieurs valeurs (ou expressions, dans le jargon de la programmation) et qui retourne une autre valeur (donc la construction elle-même devient une expression).

Vous trouverez dans la suite de ce chapitre des tableaux présentant quelques séries d'opérateurs qu'il est indispensable de connaître.

Description	Symbole	Exemple
Renvoie "true" si les valeurs \$a et \$b sont égales	Egaux (=)	\$a == \$b
Renvoie "true" si les valeurs \$a et \$b sont égales et de même type	Identiques (==>)	\$a === \$b
Renvoie "true" si les valeurs \$a et \$b sont différentes	Différents (<>)	\$a != \$b or \$a <> \$b
Renvoie "true" si les valeurs \$a et \$b sont différentes, ou de type différent	Pas identiques (!==)	\$a !== \$b
Renvoie "true" si la première valeur est inférieure à la seconde	Inférieur à (<)	\$a < \$b
Renvoie "true" si la première valeur est supérieure à la seconde	Supérieur à (>)	\$a > \$b
Renvoie "true" si la première valeur est inférieure ou égale à la seconde	Inférieur ou égal à (<=)	\$a <= \$b
Renvoie "true" si la première valeur est supérieure ou égale à la seconde	Supérieur ou égal à (>=)	\$a >= \$b

Cette distinction entre « Egaux » et « Identiques » peut sembler étrange. Pour la comprendre, un exemple s'impose :

```
$val1 = 123;
$val2 = '123';

echo 'test1'.PHP_EOL ;
if ($val1 == $val2) {
    echo "les contenus sont similaires mais pas de même type,";
    echo "le double égal renvoie 'true' dans ce cas".PHP_EOL;
} else {
    echo "ce message ne s'affichera pas";
}

echo 'test2'.PHP_EOL ;
if ($val1 === $val2) {
    echo "ce message ne s'affichera pas";
} else {
    echo "les contenus sont similaires mais pas de même type,";
    echo "le triple égal renvoie 'false' dans ce cas".PHP_EOL;
}
```

Ce qu'il faut comprendre, c'est que dans le cas du double égal, PHP convertit la valeur alphanumérique de la variable \$val2 dans le même type que celui de la variable \$val1. Si après cette conversion, le contenu des variables \$val1 et \$val2 est bien identique, alors la condition renvoie « true ».

On peut combiner plusieurs conditions dans un même test, ce petit tableau présente les opérateurs permettant de combiner les conditions :

Description	Symbol	Exemple
Renvoie "true" si les 2 évaluations renvoient "true"	and	\$a and \$b
Retourner vrai si l'une ou l'autre évaluation est vraie ou si les deux sont vraies.	or	\$a or \$b
Retournez vrai si l'une ou l'autre évaluation est vraie, mais pas les deux.	xor	\$a xor \$b
Renvoie "true" si l'évaluation renvoie "false" (lire "non vrai")	!	!\$a
identique à "and" (et plus utilisé que « and »)	&&	\$a && \$b
identique à "or" (et plus utilisé que « or »)		\$a \$b

Quelquefois, on peut rencontrer des conditions complexes, nécessitant de définir des priorités dans les conditions, dès lors l'ajout de parenthèses peut se révéler nécessaire, voire indispensable. Amusez-vous à tester et modifier les tests ci-dessous :

```
$val1 = 123;
$val2 = '123';
if (($val1 == $val2 || $val1 == true) && $val1 === 124) {
    echo 'blablabla';
}
if ($val1 == $val2 || $val1 == true && $val1 === 124) {
    echo 'blublublublu';
}
```

Les développeurs anglophones, utilisent des abréviations qu'il est intéressant de connaître, particulièrement quand on a besoin de lire de la documentation en anglais :

Français	Anglais	Abréviation anglaise
Egal (=)	Equal	EQ
Different (<>)	Not equal	NE
Inférieur à (<)	Less Than	LT
Supérieur à (>)	Greater Than	GT
Inférieur ou égal à (<=)	Less or Equal	LE
Supérieur ou égal à (>=)	Greater or Equal	GE

5.2 Préparation d'un jeu de données SQL avec Excel

Ce chapitre présente une astuce simple pour créer rapidement une table SQL et son contenu à partir d'une liste extraite d'une page HTML.

Il s'agit en l'occurrence de créer une table contenant une liste de pays.

Cette technique simple mais peu connue permet de constituer rapidement des jeux de données pouvant être utilisés dans le cadre de tests ou de petits projets.

On peut récupérer cette liste de pays sur Wikipedia, sous la rubrique ISO_3166-1 :

http://fr.wikipedia.org/wiki/ISO_3166-1

Mais pour gagner du temps, on pourra se référer au code SQL de la table LSTPAYS fourni en annexe du présent document. Ce source permettra de créer rapidement la table des pays, puis de l'exporter sous Excel (par exemple via le logiciel System i Navigator), pour pouvoir tester ce qui va suivre.

Après avoir copié-collé le tableau des pays dans Excel, et avoir éliminé les données qui ne nous intéressaient pas, on obtient le tableau suivant :

	A	B	C	D
1	Nom français	Nom ISO	Norme	Désignation
2	AFG	AF	(ISO 3166-2)	AFGHANISTAN
3	ZAF	ZA	(ISO 3166-2)	AFRIQUE DU SUD
4	ALA	AX	(ISO 3166-2)	ALAND, ILES
5	ALB	AL	(ISO 3166-2)	ALBANIE
6	DZA	DZ	(ISO 3166-2)	ALGERIE
7	DEU	DE	(ISO 3166-2)	ALLEMAGNE
8	AND	AD	(ISO 3166-2)	ANDORRE
9	AGO	AO	(ISO 3166-2)	ANGOLA
10	AIA	AI	(ISO 3166-2)	ANGUILLA
11	ATA	AQ	(ISO 3166-2)	ANTARCTIQUE
12	ATG	AG	(ISO 3166-2)	ANTIGUA-ET-BARBUDA
13	ANT	AN	(ISO 3166-2)	ANTILLES NEERLANDAISES

Nous souhaitons récupérer les colonnes A, B et D. Pour ce faire, nous allons créer une colonne supplémentaire dans laquelle nous allons utiliser une formule de concaténation, de manière à créer des requêtes INSERT pour chaque ligne du tableau Excel. Concrètement, la requête de concaténation se présente de la façon suivante :

```
= "INSERT INTO MABIB/LSTPAYS VALUES (" & A2 & ", " & B2 & ", " & D2 & ");"
```

Dès que la requête fonctionne pour une ligne, vous pouvez la dupliquer sur toutes les lignes en-dessous.

Il me semble que c'est plus parlant quand on le voit en action sur Excel :

The screenshot shows an Excel spreadsheet titled "Microsoft Excel - liste_pays.xls". The data is organized into two columns: "Désignation" (Designation) in column D and an SQL INSERT statement in column E. The data rows are as follows:

	D	E
1	Désignation	
2	AFGHANISTAN	INSERT INTO MABIB/LSTPAYS VALUES ('AFG', 'AF', 'AFGHANISTAN');
3	AFRIQUE DU SUD	INSERT INTO MABIB/LSTPAYS VALUES ('ZAF', 'ZA', 'AFRIQUE DU SUD');
4	ALAND, ILES	INSERT INTO MABIB/LSTPAYS VALUES ('ALA', 'AX', 'ALAND, ILES');
5	ALBANIE	INSERT INTO MABIB/LSTPAYS VALUES ('ALB', 'AL', 'ALBANIE');
6	ALGERIE	INSERT INTO MABIB/LSTPAYS VALUES ('DZA', 'DZ', 'ALGERIE');
7	ALLEMAGNE	INSERT INTO MABIB/LSTPAYS VALUES ('DEU', 'DE', 'ALLEMAGNE');

Il faut ensuite copier le contenu de la colonne E dans le presse-papier, puis effectuer un « collage spécial » vers une colonne vierge d'Excel en utilisant l'option « par valeurs ». Vous obtenez ainsi un script SQL d'insertion de toutes les lignes du tableau Excel.

J'allais oublier un détail important : le problème des quotes, ou apostrophes. Je vous invite à regarder la colonne E se situant en face de la ligne de la « COTE D'IVOIRE ». Normalement, vous devriez avoir ceci :

```
INSERT INTO MABIB.LSTPAYS VALUES ('CIV', 'CI', 'COTE D'IVOIRE');
```

Si vous y regardez de près, vous constaterez que vous avez un nombre d'apostrophes impair entre les parenthèses du mot-clé SQL VALUES. Donc cette requête SQL ne pourra pas fonctionner, pas plus sous DB2 que sous MySQL. Et on retrouve le même problème sur tous les libellés contenant une ou plusieurs apostrophes.

Vous pouvez corriger facilement le problème en utilisant la fonction Excel suivante :

```
=SUBSTITUE(D2;""";"""")
```

... ce qui nous donne la formule Excel suivante :

```
="INSERT INTO MABIB/LSTPAYS VALUES ('" & A2 & '", "' & B2 & '", "' & SUBSTITUE(D2;""";"""") & "');"
```

Après correction, la requête d'insertion pour la Côte d'Ivoire ressemblera à ceci :

```
INSERT INTO MABIB.LSTPAYS VALUES ('CIV', 'CI', 'COTE D'IVOIRE');
```

La correction étant effectuée pour l'ensemble des lignes du tableau, vous pouvez recopier la colonne E dans le presse-papier Windows, puis effectuer un « collage spécial par valeur » dans une colonne vierge d'une autre feuille Excel.

Vous disposez maintenant d'un script SQL d'insertion prêt à l'emploi vous permettant d'alimenter la table SQL des pays, que vous pouvez sauvegarder au format texte.

Pour gagner du temps, vous trouverez au chapitre suivant une table des pays déjà préparée, qui pourra servir pour le développement de tests.

5.3 Tables des pays au format SQL

La table SQL ci-dessous pourra nous servir de table exemple pour tester différentes techniques en PHP.

```
CREATE TABLE `countries` (
  `id` int(6) NOT NULL auto_increment,
  `codinter` char(3) NOT NULL default '',
  `codfra` char(2) NOT NULL default '',
  `countryname` varchar(250) NOT NULL default '',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

INSERT INTO countries (codinter, codfra, countryname)
VALUES
  ('AFG', 'AF', 'AFGHANISTAN'),
  ('ZAF', 'ZA', 'AFRIQUE DU SUD'),
  ('ALA', 'AX', 'ALAND, ILES'),
  ('ALB', 'AL', 'ALBANIE'),
  ('DZA', 'DZ', 'ALGERIE'),
  ('DEU', 'DE', 'ALLEMAGNE'),
  ('AND', 'AD', 'ANDORRE'),
  ('AGO', 'AO', 'ANGOLA'),
  ('AIA', 'AI', 'ANGUILLA'),
  ('ATA', 'AQ', 'ANTARCTIQUE'),
  ('ATG', 'AG', 'ANTIGUA-ET-BARBUDA'),
  ('ANT', 'AN', 'ANTILLES NEERLANDAISES'),
  ('SAU', 'SA', 'ARABIE SAOUDITE'),
  ('ARG', 'AR', 'ARGENTINE'),
  ('ARM', 'AM', 'ARMENIE'),
  ('ABW', 'AW', 'ARUBA'),
  ('AUS', 'AU', 'AUSTRALIE'),
  ('AUT', 'AT', 'AUTRICHE'),
  ('AZE', 'AZ', 'AZERBAIDJAN'),
  ('BHS', 'BS', 'BAHAMAS'),
  ('BHR', 'BH', 'BAHREIN'),
  ('BGD', 'BD', 'BANGLADESH'),
  ('BRB', 'BB', 'BARBADE'),
  ('BLR', 'BY', 'BELARUS'),
  ('BEL', 'BE', 'BELGIQUE'),
  ('BLZ', 'BZ', 'BELIZE'),
  ('BEN', 'BJ', 'BENIN'),
```

```
('BMU', 'BM', 'BERMUDES'),
('BTN', 'BT', 'BHOUTAN'),
('BOL', 'BO', 'BOLIVIE'),
('BIH', 'BA', 'BOSNIE-HERZEGOVINE'),
('BWA', 'BW', 'BOTSWANA'),
('BVT', 'BV', 'BOUVET, ILE'),
('BRA', 'BR', 'BRESIL'),
('BRN', 'BN', 'BRUNEI DARUSSALAM'),
('BGR', 'BG', 'BULGARIE'),
('BFA', 'BF', 'BURKINA FASO'),
('BDI', 'BI', 'BURUNDI'),
('CYM', 'KY', 'CAIMANES, ILES'),
('KHM', 'KH', 'CAMBODGE'),
('CMR', 'CM', 'CAMEROUN'),
('CAN', 'CA', 'CANADA'),
('CPV', 'CV', 'CAP-VERT'),
('CAF', 'CF', 'CENTRAFRICAINE, REPUBLIQUE'),
('CHL', 'CL', 'CHILI'),
('CHN', 'CN', 'CHINE'),
('CXR', 'CX', 'CHRISTMAS, ILE'),
('CYP', 'CY', 'CHYPRE'),
('CCK', 'CC', 'COCOS (KEELING), ILES'),
('COL', 'CO', 'COLOMBIE'),
('COM', 'KM', 'COMORES'),
('COG', 'CG', 'CONGO'),
('COD', 'CD',
    'CONGO, LA REPUBLIQUE DEMOCRATIQUE DU'),
('COK', 'CK', 'COOK, ILES'),
('KOR', 'KR', 'COREE, REPUBLIQUE DE'),
('PRK', 'KP',
    'COREE, REPUBLIQUE POPULAIRE DEMOCRATIQUE DE'),
('CRI', 'CR', 'COSTA RICA'),
('CIV', 'CI', 'COTE D'IVOIRE'),
('HRV', 'HR', 'CROATIE'),
('CUB', 'CU', 'CUBA'),
('DNK', 'DK', 'DANEMARK'),
('DJI', 'DJ', 'DJIBOUTI'),
('DOM', 'DO', 'DOMINICAINE, REPUBLIQUE'),
('DMA', 'DM', 'DOMINIQUE'),
('EGY', 'EG', 'EGYPTE'),
('SLV', 'SV', 'EL SALVADOR'),
('ARE', 'AE', 'EMIRATS ARABES UNIS'),
('ECU', 'EC', 'EQUATEUR'),
('ERI', 'ER', 'ERYTHREE'),
('ESP', 'ES', 'ESPAGNE'),
('EST', 'EE', 'ESTONIE'),
('USA', 'US', 'Etats-Unis'),
('ETH', 'ET', 'ETHIOPIE'),
('FLK', 'FK', 'FALKLAND, ILES (MALVINAS)'),
('FRO', 'FO', 'FEROE, ILES'),
('FJI', 'FJ', 'FIDJI'),
('FIN', 'FI', 'FINLANDE'),
('FRA', 'FR', 'FRANCE'),
('GAB', 'GA', 'GABON'),
```

```

('GMB ', 'GM ', 'GAMBIE'),
('GEO ', 'GE ', 'GEORGIE'),
('SGS ', 'GS ',
    'GEORGIE DU SUD ET LES ILES SANDWICH DU SUD'),
('GHA ', 'GH ', 'GHANA'),
('GIB ', 'GI ', 'GIBRALTAR'),
('GRC ', 'GR ', 'GRECE '),
('GRD ', 'GD ', 'GRENADE'),
('GRL ', 'GL ', 'GROENLAND'),
('GLP ', 'GP ', 'GUADELOUPE'),
('GUM ', 'GU ', 'GUAM'),
('GTM ', 'GT ', 'GUATEMALA'),
('GGY ', 'GG ', 'GUERNESEY'),
('GIN ', 'GN ', 'GUINEE '),
('GNB ', 'GW ', 'GUINEE BISSAU '),
('GNQ ', 'GQ ', 'GUINEE EQUATORIALE '),
('GUY ', 'GY ', 'GUYANA'),
('GUF ', 'GF ', 'GUYANE FRANCAISE '),
('HTI ', 'HT ', 'HAITI '),
('HMD ', 'HM ',
    'HEARD, ILE ET MCDONALD, ILES'),
('HND ', 'HN ', 'HONDURAS'),
('HKG ', 'HK ', 'HONG KONG'),
('HUN ', 'HU ', 'HONGRIE'),
('IMN ', 'IM ', 'ILE DE MAN '),
('UMI ', 'UM ',
    'ILES MINEURES ELOIGNEES DES ETATS-UNIS '),
('VGB ', 'VG ',
    'ILES VIERGES BRITANNIQUES'),
('VIR ', 'VI ',
    'ILES VIERGES DES ETATS-UNIS '),
('IND ', 'IN ', 'INDE'),
('IDN ', 'ID ', 'INDONESIE '),
('IRN ', 'IR ',
    'IRAN, REPUBLIQUE ISLAMIQUE D'' '),
('IRQ ', 'IQ ', 'IRAQ'),
('IRL ', 'IE ', 'IRLANDE '),
('ISL ', 'IS ', 'ISLANDE '),
('ISR ', 'IL ', 'ISRAEL '),
('ITA ', 'IT ', 'ITALIE '),
('JAM ', 'JM ', 'JAMAIQUE '),
('JPN ', 'JP ', 'JAPON '),
('JEY ', 'JE ', 'JERSEY '),
('JOR ', 'JO ', 'JORDANIE '),
('KAZ ', 'KZ ', 'KAZAKHSTAN '),
('KEN ', 'KE ', 'KENYA '),
('KGZ ', 'KG ', 'KIRGHIZISTAN '),
('KIR ', 'KI ', 'KIRIBATI '),
('KWT ', 'KW ', 'KOWEIT '),
('LAO ', 'LA ',
    'LAOS, REPUBLIQUE DEMOCRATIQUE POPULAIRE '),
('LSO ', 'LS ', 'LESOTHO '),
('LVA ', 'LV ', 'LETTONIE '),
('LBN ', 'LB ', 'LIBAN ')

```

```

('LBR', 'LR', 'LIBERIA'),
('LBY', 'LY',
    'LIBYENNE, JAMAHIRIYA ARABE'),
('LIE', 'LI', 'LIECHTENSTEIN'),
('LTU', 'LT', 'LITUANIE'),
('LUX', 'LU', 'LUXEMBOURG'),
('MAC', 'MO', 'MACAO'),
('MKD', 'MK',
    'MACEDOINE, L''EX-REPUBLIQUE YOUGOSLAVE DE '),
('MDG', 'MG', 'MADAGASCAR'),
('MYS', 'MY', 'MALAISIE'),
('MWI', 'MW', 'MALAWI'),
('MDV', 'MV', 'MALDIVES'),
('MLI', 'ML', 'MALI'),
('MLT', 'MT', 'MALTE'),
('MNP', 'MP',
    'MARIANNES DU NORD, ILES'),
('MAR', 'MA', 'MAROC'),
('MHL', 'MH', 'MARSHALL, ILES'),
('MTQ', 'MQ', 'MARTINIQUE'),
('MUS', 'MU', 'MAURICE'),
('MRT', 'MR', 'MAURITANIE'),
('MYT', 'YT', 'MAYOTTE'),
('MEX', 'MX', 'MEXIQUE'),
('FSM', 'FM',
    'MICRONESIE, ETATS FEDERES DE '),
('MDA', 'MD', 'MOLDOVA'),
('MCO', 'MC', 'MONACO'),
('MNG', 'MN', 'MONGOLIE'),
('MNE', 'ME', 'MONTENEGRO'),
('MSR', 'MS', 'MONTSERRAT'),
('MOZ', 'MZ', 'MOZAMBIQUE'),
('MMR', 'MM', 'MYANMAR'),
('NAM', 'NA', 'NAMIBIE'),
('NRU', 'NR', 'NAURU'),
('NPL', 'NP', 'NEPAL '),
('NIC', 'NI', 'NICARAGUA'),
('NER', 'NE', 'NIGER'),
('NGA', 'NG', 'NIGERIA '),
('NIU', 'NU', 'NIUE '),
('NFK', 'NF', 'NORFOLK, ILE '),
('NOR', 'NO', 'NORVEGE '),
('NCL', 'NC', 'NOUVELLE-CALEDONIE'),
('NZL', 'NZ', 'NOUVELLE-ZELANDE '),
('IOT', 'IO',
    'OCEAN INDIEN, TERRITOIRE BRITANNIQUE DE L'' '),
('OMN', 'OM', 'OMAN'),
('UGA', 'UG', 'OUGANDA'),
('UZB', 'UZ', 'OUZBEKISTAN '),
('PAK', 'PK', 'PAKISTAN'),
('PLW', 'PW', 'PALAOS'),
('PSE', 'PS',
    'PALESTINIEN OCCUPE, TERRITOIRE'),
('PAN', 'PA', 'PANAMA'),

```

```
('PNG', 'PG',
    'PAPOUASIE-NOUVELLE-GUINEE'),
('PRY', 'PY',
    'PARAGUAY'),
('NLD', 'NL',
    'PAYS-BAS'),
('PER', 'PE',
    'PEROU'),
('PHL', 'PH',
    'PHILIPPINES'),
('PCN', 'PN',
    'PITCAIRN'),
('POL', 'PL',
    'POLOGNE'),
('PYF', 'PF',
    'POLYNESIE FRANCAISE'),
('PRI', 'PR',
    'PORTO RICO'),
('PRT', 'PT',
    'PORTUGAL'),
('QAT', 'QA',
    'QATAR'),
('REU', 'RE',
    'REUNION'),
('ROU', 'RO',
    'ROUMANIE'),
('GBR', 'GB',
    'ROYAUME-UNI'),
('RUS', 'RU',
    'RUSSIE, FEDERATION DE'),
('RWA', 'RW',
    'RWANDA'),
('ESH', 'EH',
    'SAHARA OCCIDENTAL'),
('BLM', 'BL',
    'SAINT-BARTHELEMY'),
('KNA', 'KN',
    'SAINT-KITTS-ET-NEVIS'),
('SMR', 'SM',
    'SAINT-MARIN'),
('MAF', 'MF',
    'SAINT-MARTIN (PARTIE FRANCAISE)'),
('SPM', 'PM',
    'SAINT-PIERRE-ET-MIQUELON'),
('VAT', 'VA',
    'SAINT-SIEGE (ETAT DE LA CITE DU VATICAN)'),
('VCT', 'VC',
    'SAINT-VINCENT-ET-LES GRENADINES'),
('SHN', 'SH',
    'SAINTE-HELENE'),
('LCA', 'LC',
    'SAINTE-LUCIE'),
('SLB', 'SB',
    'SALOMON, ILES'),
('WSM', 'WS',
    'SAMOA'),
('ASM', 'AS',
    'SAMOA AMERICAINES'),
('STP', 'ST',
    'SAO TOME-ET-PRINCIPE'),
('SEN', 'SN',
    'SENEGAL'),
('SRB', 'RS',
    'SERBIE'),
('SYC', 'SC',
    'SEYCHELLES'),
('SLE', 'SL',
    'SIERRA LEONE'),
('SGP', 'SG',
    'SINGAPOUR'),
('SVK', 'SK',
    'SLOVAQUIE'),
('SVN', 'SI',
    'SLOVENIE'),
('SOM', 'SO',
    'SOMALIE'),
('SDN', 'SD',
    'SOUDAN'),
('LKA', 'LK',
    'SRI LANKA'),
('SWE', 'SE',
    'SUEDE'),
('CHE', 'CH',
    'SUISSE'),
('SUR', 'SR',
    'SURINAME'),
('SJM', 'SJ',
    'SVALBARD ET ILE JAN MAYEN'),
('SWZ', 'SZ',
    'SWAZILAND'),
('SYR', 'SY',
    'SYRIENNE, REPUBLIQUE ARABE'),
('TJK', 'TJ',
    'TADJIKISTAN'),
```

```
('TWN', 'TW', 'TAIWAN, PROVINCE DE CHINE'),  
('TZA', 'TZ', 'TANZANIE, REPUBLIQUE UNIE DE'),  
('TCD', 'TD', 'TCHAD'),  
('CZE', 'CZ', 'TCHEQUE, REPUBLIQUE'),  
('ATF', 'TF', 'TERRES AUSTRALES FRANCAISES'),  
('THA', 'TH', 'THAILANDE'),  
('TLS', 'TL', 'TIMOR-LESTE'),  
('TGO', 'TG', 'TOGO'),  
('TKL', 'TK', 'TOKELAU'),  
('TON', 'TO', 'TONGA'),  
('TTO', 'TT', 'TRINITE-ET-TOBAGO'),  
('TUN', 'TN', 'TUNISIE'),  
('TKM', 'TM', 'TURKMENISTAN'),  
('TCA', 'TC', 'TURKS ET CAIQUES, ILES'),  
('TUR', 'TR', 'TURQUIE'),  
('TUV', 'TV', 'TUVALU'),  
('UKR', 'UA', 'UKRAINE'),  
('URY', 'UY', 'URUGUAY'),  
('VUT', 'VU', 'VANUATU'),  
('VEN', 'VE', 'VENEZUELA'),  
('VNM', 'VN', 'VIET NAM'),  
('WLF', 'WF', 'WALLIS-ET-FUTUNA'),  
('YEM', 'YE', 'YEMEN'),  
('ZMB', 'ZM', 'ZAMBIE'),  
('ZWE', 'ZW', 'ZIMBABWE');
```

5.4 Mesure de performances

Il est quelquefois utile de pouvoir mesurer les performances d'un script PHP.

On peut pour ce faire s'appuyer sur la fonction PHP microtime() que l'on encapsulera dans une fonction getmicrotime(), de la façon suivante :

```
function getmicrotime() {  
    list($usec, $sec) = explode(" ", microtime());  
    return ((float)$usec + (float)$sec);  
  
}  
  
// premier appel de la fonction getmicrotime() avant l'exécution du code à "mesurer"  
$time_start = getmicrotime();  
  
// placer ici le code dont vous souhaitez mesurer les performances  
  
// second et dernier appel de la fonction getmicrotime()  
$time_stop = getmicrotime();  
$time_dif = $time_stop - $time_start;  
  
echo "<p><h2>Temps d'exécution de la reprise :</h2> {$time_dif} secondes </p>"  
;
```

5.5 Les filtres

L'extension "Filter" apporte à PHP la possibilité de filtrer les données soit en les validant, soit en les nettoyant.

Cette extension est particulièrement utile pour contrôler et filtrer les données provenant de l'extérieur comme des données de formulaires par exemple.

Il y a donc deux possibilités de filtrage : la validation et le nettoyage.

La Validation sert à vérifier si une donnée répond à certains critères (comme par exemple le filtre de validation d'email : FILTER_VALIDATE_EMAIL). En cas d'anomalie, un message d'anomalie est renvoyé par la fonction filter_var(), mais la donnée contrôlée n'est pas modifiée.

Le nettoyage va nettoyer les données, par exemple en retirant des caractères indésirables. Par exemple, la fonction filter_var() associée au mot clé FILTER_SANITIZE_EMAIL va permettre d'éliminer les caractères inappropriés pour une adresse email.

Lien vers la documentation officielle relative à l'extension "Filter" :

<http://www.php.net/manual/fr/intro.filter.php>

A voir aussi dans la documentation officielle :

- la page consacrée aux autres filtres :

<http://www.php.net/manual/fr/filter.filters.misc.php>

- la page consacrée aux "drapeaux" des filtres :

<http://www.php.net/manual/fr/filter.filters.flags.php>

Quelques exemples ci-dessous pris sur :

<http://www.php.net/manual/fr/filter.examples.validation.php>

```
/*
 * Exemple #1 Validation d'adresses email avec filter_var()
 */
$email_a = 'joe@example.com';
$email_b = 'bogus';

if (filter_var($email_a, FILTER_VALIDATE_EMAIL)) {
```

```

        echo "Cette ($email_a) adresse email est considérée comme valide.";
    }
    if (filter_var($email_b, FILTER_VALIDATE_EMAIL)) {
        echo "Cette ($email_b) adresse email est considérée comme valide.";
    }

/*
 * Exemple #2 Validation d'adresses IP avec filter_var()
 */
$ip_a = '127.0.0.1';
$ip_b = '42.42';

if (filter_var($ip_a, FILTER_VALIDATE_IP)) {
    echo "Cette ($ip_a) adresse IP est considérée comme valide.";
}
if (filter_var($ip_b, FILTER_VALIDATE_IP)) {
    echo "Cette ($ip_b) adresse IP est considérée comme valide.";
}

/*
 * Exemple #3 Passage d'options à la fonction filter_var()
 */
$int_a = '1';
$int_b = '-1';
$int_c = '4';
$options = array(
    'options' => array(
        'min_range' => 0,
        'max_range' => 3,
    )
);
if (filter_var($int_a, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "Cet entier ($int_a) est considéré comme valide (entre 0 et 3).\n";
}
if (filter_var($int_b, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "Cet entier ($int_b) est considéré comme valide (entre 0 et 3).\n";
}
if (filter_var($int_c, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "Cet entier ($int_c) est considéré comme valide (entre 0 et 3).\n";
}

$options['options']['default'] = 1;
if (($int_c = filter_var($int_c, FILTER_VALIDATE_INT, $options)) !== FALSE) {
    echo "Cet entier ($int_c) est considéré comme valide (entre 0 et 3) et vaut
$int_c.";
}

/*
 * Exemple : Nettoyage et validation d'adresses email
 * Exemples pris sur
 * http://www.php.net/manual/fr/filter.examples.sanitization.php
 */

```

```
$a = 'joe@example.org';
$b = 'bogus - at - example dot org';
$c = '(bogus@example.org)';

$sanitized_a = filter_var($a, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_a, FILTER_VALIDATE_EMAIL)) {
    echo "Cette (a) adresse email nettoyée est considérée comme valide.";
}

$sanitized_b = filter_var($b, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_b, FILTER_VALIDATE_EMAIL)) {
    echo "Cette (b) adresse email nettoyée est considérée comme valide.";
} else {
    echo "Cette (b) adresse email nettoyée est considérée comme invalide.";
}

$sanitized_c = filter_var($c, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_c, FILTER_VALIDATE_EMAIL)) {
    echo "Cette (c) adresse email nettoyée est considérée comme valide.";
    echo "Avant : $c\n";
    echo "Après : $sanitized_c\n";
}
```

5.6 Les pièges du « select multiple »

Ce chapitre reprend l'exemple du formulaire tel qu'il a été traité au chapitre 4.4.3 (dans sa version finale).

L'exemple traité dans ce chapitre annexe est très proche de l'exemple initial, si ce n'est que le champ de type « select » du formulaire initial s'est vu doter de l'attribut « multiple ».

Le fait que le champ « select » fonctionne avec l'attribut « multiple » a un impact important sur le code PHP, car le tableau `$_POST['myselect']` va recevoir un tableau PHP (contenant la liste des options sélectionnées par l'utilisateur). Cela a un impact sur la manière dont les options du champ « select » sont « selected » ou pas, et sur la manière de filtrer et de transmettre le contenu du champ « myselect » au script « pageable.php ».

Mais l'attribut « multiple » pose aussi quelques difficultés qui sont dues au fait que le champ « select » doit impérativement contenir au moins une option « selected » pour que le champ fasse partie de la requête HTTP transmise au script PHP. Sans cela, le champ de saisie n'apparaît tout simplement pas dans `$_POST`, ce qui est un comportement pour le moins étrange. Il a donc été nécessaire de prendre en compte ce comportement imprévu, en particulier dans la fonction `get_form_select_selected()`.

Le script « pageable.php » est très légèrement impacté par la modification, son code source est donné en premier :

```
<?php
// démarrage du gestionnaire de session
session_start();

echo '<br>Contenu de $_SESSION affiché avec une boucle foreach<br>';
echo '<ul>' . PHP_EOL;
foreach($_SESSION as $postkey=>$postval) {
    $postkey = htmlentities($postkey, ENT_QUOTES, "UTF-8");
    if (is_array($postval)) {
        $tmp = [];
        foreach($postval as $postval2) {
            $tmp[] = htmlentities($postval2, ENT_QUOTES, "UTF-8");
        }
        $postval = implode(' | ', $tmp);
    } else {
        $postval = htmlentities($postval, ENT_QUOTES, "UTF-8");
    }
    echo "<li>{$postkey} => {$postval} </li>" . PHP_EOL ;
}
echo '</ul>' . PHP_EOL;
```

Le code source du script « pageform.php » démarre sur la page suivante.

```

<?php
// démarrage du gestionnaire de session
session_start();
?><!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Exemple de formulaire (2)</title>
    <link rel="stylesheet" href="pageform.css">
</head>
<body>
<?php
    // Initialisation du tableau des erreurs
    $erreurs = [];
    $gooddata = [];
    // Boucle pour analyser si des erreurs sont présentes
    foreach($_POST as $keypost=>$valpost) {
        if ($keypost != 'valid') {
            if ($keypost == 'myselect') {
                // myselect est un tableau, il doit être traité à part
                if (count($valpost) == 0) {
                    $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
                } else {
                    $gooddata[$keypost] = [];
                    foreach($valpost as $valpost2) {
                        $tmp = trim(strip_tags($valpost2));
                        if ($tmp != '') {
                            $gooddata[$keypost][] = $tmp;
                        }
                    }
                    if (count($gooddata[$keypost])==0) {
                        $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
                    }
                }
            } else {
                $valpost = strip_tags($valpost);
                $valpost = trim($valpost);
                if ($valpost == '') {
                    $erreurs[$keypost] = "Champ $keypost : zone obligatoire";
                } else {
                    $gooddata[$keypost] = $valpost;
                }
            }
        }
    }
    // si le tableau $gooddata contient des données et qu'il n'y a pas d'erreurs
    if (count($gooddata)>0 && count($erreurs)==0) {
        // copie des bonnes données dans $_SESSION
        $_SESSION = $gooddata;
        // redirection vers "pagecible.php"
        header('location: pagecible.php');
        exit; // on stoppe l'exécution du script courant
    }
    // Si des erreurs sont présentes, alors on en affiche la liste
    if (count($erreurs)>0) {
        echo '<fieldset style="background-color: bisque; border-radius: 5px;">' . PHP_EOL;
        echo '<legend>Liste des erreurs</legend>' . PHP_EOL;
        echo '<ul>' . PHP_EOL;
        foreach($erreurs as $valerr) {
            echo "<li>$valerr</li>" . PHP_EOL;
        }
    }

```

```

        echo '</ul>'.PHP_EOL;
        echo '</fieldset>'.PHP_EOL;
    }

    /**
     * Renvoie en sortie la valeur d'un champ de saisie
     * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
     *   'post' (valeur par défaut)
     *   'get'
     * @param $field
     * @param string $method
     * @return string
     */
    function get_form_input_value($field, $method='post') {
        // suppression des blancs parasites et forçage des valeurs en minuscule
        $field = strtolower(trim($field));
        $method = strtolower(trim($method));

        if ($method == 'get') {
            $value = isset($_GET[$field]) ? $_GET[$field] : '';
        } else {
            $value = isset($_POST[$field]) ? $_POST[$field] : '';
        }
        $value = trim(strip_tags($value));
        $value = htmlentities($value, ENT_QUOTES, "UTF-8");
        return $value;
    }

    /**
     * Renvoie en sortie la valeur "selected" pour les champs de type SELECT
     * si la valeur du champ considéré est égale au second paramètre
     * Le paramètre $method est facultatif et peut contenir les 2 valeurs suivantes :
     *   'post' (valeur par défaut)
     *   'get'
     * Le paramètre $multiple est facultatif (booléen à false par défaut)
     * @param $field
     * @param $option_key
     * @param string $method
     * @param string $multiple
     * @return string
     */
    function get_form_select_selected($field, $option_key, $method='post',
        $multiple=false) {
        // suppression des blancs parasites et forçage des valeurs en minuscule
        $field = strtolower(trim($field));
        $method = strtolower(trim($method));
        if ($method == 'get') {
            $value = isset($_GET[$field]) ? $_GET[$field] : '';
        } else {
            $value = isset($_POST[$field]) ? $_POST[$field] : '';
        }
        if ($multiple == false) {
            $value = trim($value);
            if ($value == $option_key) {
                return 'selected';
            }
        } else {
            if (!is_array($value) || count($value)==0) {
                // si le tableau est vide et que $option_key est à blanc,

```

```

        // cela signifie qu'on est sur la première option et il est
        // préférable de mettre cette option à "selected" pour bloquer
        // un comportement étrange constaté sur Firefox et Chrome, avec
        // les champs "select" dotés de l'attribut "multiple"
        if ($option_key == '') {
            return 'selected';
        }
        // dans tous les autres cas, on sort sans le "selected"
        return '';
    } else {
        if (in_array($option_key, $value)) {
            // si l'option se trouve dans le tableau alors sortie avec
"selected"
            return 'selected';
        }
    }
    return '';
}
?>
<form method="POST">
    <fieldset>
        <legend>Formulaire en vrac</legend>
        <p>
            <label for="search_field">Exemple de champ "search"</label>
            <input type="search" name="search" id="search_field"
                placeholder="saisissez un critère de recherche"
                value=<?php echo get_form_input_value('search'); ?>>
        />
    </p>
    <p>
        <label for="number_field">Exemple de champ "number"</label>
        <input type="number" name="number" id="number_field"
                value=<?php echo get_form_input_value('number'); ?>>
    />
    </p>
    <p>
        <label for="select_field">Exemple de champ "select"</label>
        <select name="myselect[]" id="select_field" multiple size="4">
            <?php
                // la première option (avec code à blanc) est intégrée dans la boucle
                // afin de pouvoir être sélectionnée par défaut si aucune option
                // n'est sélectionnée
                foreach(array('', 'A', 'B', 'C') as $valselect) {
                    if ($valselect == '') {
                        echo "<option value=\"{$valselect}\\" .
                            . get_form_select_selected('myselect', $valselect, 'post',
true)
                            . ">Choisissez une valeur</option>" . PHP_EOL;
                    } else {
                        echo "<option value=\"{$valselect}\\" .
                            . get_form_select_selected('myselect', $valselect, 'post',
true)
                            . ">valeur {$valselect}</option>" . PHP_EOL;
                    }
                };
            ?>
        </select>
    </p>
</fieldset>

```

```
<input type="submit" name="valid" value="Valider" />
</form>
</body>
</html>
```

5.7 Une classe DBWrapper pour MySQL

Code source de la classe DBWrapper :

```
<?php
/*
 * Interface destinée à renforcer la fiabilité de la classe utilisatrice
 */
interface intDBWrapper {

    static function selectOne($db, $sql, $args = array(), $fetch_mode_num = false);

    static function selectBlock($db, $sql, $args = array());

    static function execute($db, $sql, $args = array());

    static function getLastInsertId($db);

}

/**
 * Classe DBWrapper destinée à simplifier l'utilisation de requêtes SQL
 * au sein de l'application
 */
abstract class DBWrapper implements intDBWrapper {

    final public function __construct() {
        throw new Exception("Classe statique - instantiation impossible.");
    }

    /**
     * Méthode destinées à renvoyer un jeu de données d'une seule ligne
     * renvoyée sous forme d'un tableau associatif (par défaut) ou d'un
     * tableau indicé numériquement (paramètre 4 = true)
     * @param PDO $db
     * @param string $sql
     * @param array $args
     * @param boolean $fetch_mode_num
     * @return boolean
     */
    public static function selectOne($db, $sql, $args = array(),
                                    $fetch_mode_num = false) {
        $rows = array();
        $result = array();
        // $sql .= ' limit 1';
        if (!is_array($args)) {
            if (trim($args) != '') {
                $args = array($args);
            } else {
                $args = array();
            }
        }
        try {
            $st = $db->prepare($sql);
            $ok = $st->execute($args);
            if ($ok) {

```

```

/*
 * par défaut c'est le mode "fetch array" qui est utilisé,
 * mais dans certains cas le mode "fetch column" peut être utile
 * notamment quand on ne souhaite récupérer qu'une seule colonne
 */
if ($fetch_mode_num === true) {
    $result = $st->fetch(PDO::FETCH_NUM);
} else {
    $result = $st->fetch(PDO::FETCH_ASSOC);
}
} else {
    $result = false;
}
unset($st);
return $result;
} catch (PDOException $e) {
    self::DbWrapperException($e, $sql, $args);
} catch (Exception $e) {
    self::DbWrapperException($e, $sql, $args);
}

/**
 * Méthode destinée à exécuter des requêtes renvoyant un jeu de
 * données (en anglais "dataset") de plusieurs lignes, sous forme
 * de tableau associatif
 * @param PDO $db
 * @param string $sql
 * @param array $args
 * @return boolean
 */
public static function selectBlock($db, $sql, $args = array()) {
    $rows = array();
    if (!is_array($args)) {
        if (trim($args) != '') {
            $args = array($args);
        } else {
            $args = array();
        }
    }
    try {
        $st = $db->prepare($sql);
        $ok = $st->execute($args);
        if ($ok) {
            $row = $st->fetch(PDO::FETCH_ASSOC);
            while ($row != false) {
                $rows [] = $row;
                $row = $st->fetch(PDO::FETCH_ASSOC);
            }
        } else {
            $rows = false;
        }
        unset($st);
        return $rows;
    } catch (PDOException $e) {
        self::DbWrapperException($e, $sql, $args);
    } catch (Exception $e) {
        self::DbWrapperException($e, $sql, $args);
    }
}
}

```

```

    /**
 * Méthode permettant d'exécuter des requêtes de type INSERT, UPDATE
 * ou DELETE.
 * Renvoie false si la requête s'est mal passée, ou le nombre de lignes
 * traitées si l'exécution de la requête SQL s'est bien passée
 * @param PDO $db
 * @param string $sql
 * @param array $args
 * @return int
 */
public static function execute($db, $sql, $args = array()) {
    if (!is_array($args)) {
        if (trim($args) != '') {
            $args = array($args);
        } else {
            $args = array();
        }
    }
    try {
        $st = $db->prepare($sql);
        $ok = $st->execute($args);
        if ($ok) {
            $nbrows = $st->rowCount();
        } else {
            $nbrows = 0;
        }
        return $nbrows;
    } catch (PDOException $e) {
        self::DbWrapperException($e, $sql, $args);
    } catch (Exception $e) {
        self::DbWrapperException($e, $sql, $args);
    }
    return false;
}

/**
 * Méthode permettant de récupérer le dernier ID créé dans la BD
 * (ou false si la fonction MySQL Last_Insert_ID() n'est pas en mesure
 * de renvoyer l'information (suite à une erreur par exemple)
 * @param PDO $db
 * @return boolean
 */
public static function getLastInsertId($db) {
    $sql = "SELECT last_Insert_Id() AS lastid";
    $data = self::selectOne($db, $sql, array(), true);
    if (is_array($data) && isset($data[0])) {
        return $data[0];
    } else {
        return false;
    }
}

/**
 * Méthode statique permettant un meilleur contrôle sur la présentation
 * des erreurs renvoyées dans la log PHP par cette classe
 * @param type $exc
 * @param type $sql
 * @param type $args
 */

```

```
private static function DBWrapperException($exc, $sql, $args) {
    $tab_log = [];
    $tab_log ['ErrMsg'] = $exc->getMessage();
    $tab_log ['Trace'] = $exc->getTraceAsString();
    $tab_log ['Code'] = $exc->getCode();
    $tab_log ['File'] = $exc->getFile();
    $tab_log ['Line'] = $exc->getLine();
    $tab_log ['SQL_query'] = $sql;
    $tab_log ['SQL_args'] = var_export($args, true);
    foreach($tab_log as $key=>$value) {
        error_log($key . ' => ' . $value);
    }
}
```

5.8 Jeux de données du chapitre 4.9.3

Le jeu de données ci-dessous est disponible sous forme de fichier SQL distinct, en accompagnement du présent support de cours, dans le dépôt Github suivant :

<https://github.com/gregja/PHPCorner/>

```

CREATE TABLE prd_activite (
    id int(10) unsigned NOT NULL AUTO_INCREMENT,
    nom varchar(80) DEFAULT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ;

CREATE TABLE prd_famille (
    id int(10) unsigned NOT NULL AUTO_INCREMENT,
    nom varchar(80) DEFAULT NULL,
    prd_activite_id int(10) unsigned NOT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ;

CREATE TABLE prd_produit (
    id int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    codeproduit char(50) NOT NULL,
    prd_famille_id int(10) UNSIGNED NOT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE prd_prixstd (
    id int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    prd_produit_id int(10) UNSIGNED NOT NULL,
    date_deb date NOT NULL,
    prix decimal(11,5) NOT NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- 
-- Données de la table prd_activite
-- 
INSERT INTO prd_activite (id, nom) VALUES
(1, 'Motoculture'),
(2, 'Remorques'),
(3, 'Quads'),
(4, 'Cycles'),
(5, 'Matériel agricole et espace vert'),
(6, 'Matériel de travaux publics');

-- 
-- Données de la table prd_famille
-- 
INSERT INTO prd_famille (id, nom, prd_activite_id) VALUES
(1, 'Aspirateurs électriques', 1),
(2, 'Atomiseurs', 1),
(3, 'Broyeurs', 1),
(4, 'Coupe-bordures, coupe-herbes', 1),
(5, 'Débroussailleuses', 1),
(6, 'Débroussailleuses à dos', 1),
(7, 'Découpeuses à disque', 1),
(8, 'Démousseurs', 1),
(9, 'Déneigeuses', 1),
(10, 'Motobineuses', 1),
(11, 'Motoculteurs', 1),
(12, 'Motofaucheuses', 1),
(13, 'Nettoyeurs haute pression', 1),
(14, 'Outilage à main pour le jardin', 1),

```

```

(15, 'Outilillage à main pour la forêt', 1),
(16, 'Récolteuses d'olives', 1),
(17, 'Scarificateurs', 1),
(18, 'Souffleurs à dos', 1),
(19, 'Souffleurs, aspiro-broyeurs', 1),
(20, 'Tailles-haies', 1),
(21, 'Tailles-haies sur perche', 1),
(22, 'Tarrières', 1),
(23, 'Tondeuses débroussailleuses', 1),
(24, 'Tondeuses électriques', 1),
(25, 'Tondeuses thermiques', 1),
(26, 'Tracteurs de pelouses', 1),
(27, 'Tronçonneuses à chaîne', 1),
(28, 'Tronçonneuses d'élagage', 1),
(29, 'Tronçonneuses sur perche', 1),
(30, 'Accessoires', 2),
(31, 'Agricoles', 2),
(34, 'Utilitaires', 2),
(36, 'Sports (homologués route)', 3),
(37, 'Utilitaires (homologués route)', 3),
(40, 'VTT', 4),
(41, 'Tracteurs', 5),
(42, 'Chargeuse', 6),
(43, 'Compactage', 6),
(44, 'Compresseur', 6),
(45, 'Groupe électrogène monophase', 6),
(46, 'Groupe électrogène triphasé', 6),
(47, 'Mini-pelle', 6),
(49, 'Transporteur', 6),
(56, 'Pulvérisateur', 1),
(61, 'Tondeuses robot', 1);

-- Données de la table prd_produit
--

INSERT INTO prd_produit (id, codeproduit, prd_famille_id) VALUES
(1, 'MS 290', 27),
(2, 'MS 310', 27),
(3, 'FS 350', 5),
(4, 'FS 400', 5),
(5, 'FS 450', 5),
(6, 'FS 450-K', 5),
(7, 'HS 45', 20),
(8, 'TS 400', 7),
(9, 'ME 360', 24),
(10, 'ME 400', 24),
(11, 'GE 103', 3),
(12, 'MS 200', 27),
(13, 'MS 200 T', 28),
(19, 'MS 260', 27),
(20, 'MS 260 C-B', 27),
(21, 'MS 440', 27),
(22, 'MS 460', 27),
(23, 'MS 460-R Intervention', 27),
(24, 'VH 400', 10),
(25, 'GE 35 L', 3),
(26, 'GE 105', 3),
(27, 'BT 45 Tarière', 22),
(28, 'MS 270', 27),
(29, 'MS 270 C-B', 27),
(30, 'HTE 60', 29),
(31, 'BT 45 Perceuse à bois', 22),
(32, 'FS 100 R', 5),
(33, 'FS 100', 5),
(34, 'MS 660', 27),

```

(35, 'GB 370', 3),
(36, 'GB 370 S', 3),
(37, 'MB 2 R', 25),
(38, 'MM 55', 8),
(39, 'MS 230', 27),
(40, 'MS 250', 27),
(41, 'MS 341', 27),
(42, 'MS 361', 27),
(43, 'MS 880', 27),
(44, 'FS 250', 5),
(45, 'FS 500', 5),
(46, 'FS 550', 5),
(47, 'FR 350', 6),
(48, 'FR 450', 6),
(49, 'HT 100', 29),
(50, 'HT 101', 29),
(51, 'RE 361', 13),
(52, 'RE 361 PLUS', 13),
(53, 'RE 471 PLUS', 13),
(54, 'ME 340', 24),
(55, 'GE 150', 3),
(56, 'GE 250', 3),
(57, 'VH 440', 10),
(58, 'VH 540', 10),
(59, 'VH 660', 10),
(60, 'RE 142', 13),
(61, 'RE 142 PLUS', 13),
(62, 'RE 162 PLUS', 13),
(63, 'MSE 220 C-Q', 27),
(64, 'MB 6 RH', 23),
(65, 'MSE 140 C-BQ', 27),
(66, 'MSE 160 C-BQ', 27),
(67, 'MS 192 T', 28),
(68, 'MS 230 C-BE', 27),
(69, 'MS 250 C-BE', 27),
(70, 'MS 650', 27),
(71, 'FS 130', 5),
(72, 'TS 700', 7),
(73, 'KM 100 R', 5),
(74, 'RE 551 PLUS', 13),
(75, 'RE 581', 13),
(76, 'RE 581 PLUS', 13),
(77, 'RE 661 PLUS', 13),
(78, 'RE 961 PLUS', 13),
(79, 'FS 130 R', 5),
(80, 'BR 500', 18),
(81, 'BR 550', 18),
(82, 'BR 600', 18),
(83, 'BT 121 C', 22),
(84, 'MB 650 KS', 25),
(85, 'MB 650 T', 25),
(86, 'MB 650 V', 25),
(87, 'MB 650 VS', 25),
(88, 'MB 655 KS', 25),
(89, 'MB 655 V', 25),
(90, 'MB 3 RT', 25),
(91, 'HS 81 R', 20),
(92, 'HS 86 R', 20),
(93, 'HS 86 T', 20),
(94, 'HL 100 K, 135°', 21),
(95, 'HL 100, 135°', 21),
(96, 'MB 448 T', 25),
(97, 'ME 443', 24),
(98, 'FS 90', 5),
(99, 'FS 90 R', 5),
(100, 'TS 800', 7),

```
(101, 'SE 61', 1),
(102, 'SE 61 E', 1),
(103, 'FS-KM 130', 5),
(104, 'KM 130 R', 5),
(105, 'KM 90 R', 5),
(106, 'MB 650 TK', 25),
(107, 'MB 655 GK', 25),
(108, 'MS 441', 27),
(109, 'MB 545 T', 25),
(110, 'MB 545 V', 25),
(111, 'ME 545 V', 24),
(112, 'MB 545 VE', 25),
(113, 'MB 545 VS', 25),
(114, 'MS 280 C-B I', 27),
(115, 'MS 280 I', 27),
(116, 'HT 131', 29),
(117, 'HT 130', 29),
(118, 'FS 310', 5),
(119, 'FS 87', 5),
(120, 'FS 87 R', 5),
(121, 'FR 130 T', 6),
(122, 'RB 302', 13),
(123, 'RB 402 PLUS', 13),
(124, 'RE 271', 13),
(125, 'RE 271 PLUS', 13),
(126, 'RE 281 PLUS', 13),
(127, 'MB 650 VE', 25),
(128, 'MB 755 KS', 25),
(129, 'TS 410', 7),
(130, 'TS 420', 7),
(131, 'MS 171', 27),
(132, 'MS 181', 27),
(133, 'MS 181 C-BE', 27),
(134, 'MS 211', 27),
(135, 'MS 211 C-BE', 27);
```

```
--  
-- Données de la table prd_prixstd  
--  
INSERT INTO prd_prixstd (prd_produit_id, date_deb, prix) VALUES  
(60, '2017-01-01', '459.03000'),  
(61, '2017-01-01', '500.84000'),  
(62, '2017-01-01', '584.45000'),  
(51, '2017-01-01', '1109.00000'),  
(52, '2017-01-01', '1325.00000'),  
(53, '2017-01-01', '1620.00000'),  
(75, '2017-01-01', '2390.00000'),  
(76, '2017-01-01', '2910.00000'),  
(77, '2017-01-01', '3415.00000'),  
(126, '2017-01-01', '1019.23000'),  
(124, '2017-01-01', '868.73000'),  
(125, '2017-01-01', '952.34000'),  
(101, '2017-01-01', '121.24000'),  
(78, '2017-01-01', '4300.00000'),  
(74, '2017-01-01', '2760.00000'),  
(122, '2017-01-01', '1390.00000'),  
(123, '2017-01-01', '2960.00000'),  
(50, '2017-01-01', '782.61000'),  
(49, '2017-01-01', '608.70000'),  
(117, '2017-01-01', '752.51000'),  
(116, '2017-01-01', '895.48000'),  
(80, '2017-01-01', '739.97000'),  
(81, '2017-01-01', '756.69000'),  
(82, '2017-01-01', '848.66000'),
```

```
(27, '2017-01-01', '581.10000'),  
(30, '2017-01-01', '407.19000'),  
(36, '2017-01-01', '1040.97000'),  
(11, '2017-01-01', '268.39000'),  
(26, '2017-01-01', '325.25000'),  
(56, '2017-01-01', '517.56000'),  
(55, '2017-01-01', '394.65000'),  
(10, '2017-01-01', '277.59000'),  
(24, '2017-01-01', '564.38000'),  
(57, '2017-01-01', '592.81000'),  
(58, '2017-01-01', '609.53000'),  
(59, '2017-01-01', '1045.15000'),  
(54, '2017-01-01', '166.39000'),  
(9, '2017-01-01', '186.45000'),  
(97, '2017-01-01', '243.31000'),  
(111, '2017-01-01', '460.70000'),  
(109, '2017-01-01', '542.64000'),  
(110, '2017-01-01', '584.45000'),  
(112, '2017-01-01', '776.76000'),  
(113, '2017-01-01', '760.03000'),  
(96, '2017-01-01', '469.90000'),  
(37, '2017-01-01', '333.61000'),  
(85, '2017-01-01', '677.26000'),  
(86, '2017-01-01', '739.97000'),  
(127, '2017-01-01', '932.27000'),  
(87, '2017-01-01', '835.28000'),  
(84, '2017-01-01', '985.79000'),  
(106, '2017-01-01', '819.40000'),  
(90, '2017-01-01', '606.19000'),  
(88, '2017-01-01', '1208.19000'),  
(107, '2017-01-01', '1010.87000'),  
(128, '2017-01-01', '1604.52000'),  
(64, '2017-01-01', '1024.25000');
```

5.9 PHP Manifesto

Publié par un développeur anonyme il y a quelques années, le manifeste ci-dessous a fait grand bruit dans la blogosphère PHP. Il a contribué à une prise de conscience, de la part d'une partie des développeurs PHP, de se tourner vers des solutions techniques plus "légères" que ce que proposent certains éditeurs de frameworks. La réponse apportée à ce manifeste par la communauté est sans doute à trouver du côté des micro-frameworks comme Slim, Lumen, Silex, etc...

I am a PHP developer

- I am not a Zend Framework or Symfony or CakePHP developer
- I think PHP is complicated enough

I like building small things

- I like building small things with simple purposes
- I like to make things that solve problems
- I like building small things that work together to solve larger problems

I want less code, not more

- I want to write less code, not more
- I want to manage less code, not more
- I want to support less code, not more
- I need to justify every piece of code I add to a project

I like simple, readable code

- I want to write code that is easily understood
- I want code that is easily verifiable

6. Bibliographie et Liens utiles

Commençons par une petite sélection d'ouvrages :

- *PHP & MySQL redesign*, par Luke Welling et Laura Thomson (Pearson Campupress, octobre 2012), à ma connaissance le seul livre disponible en français dans cette sélection
- *PHP Cookbook, 3rd Edition*, par David Sklar and Adam Trachtenberg (O'Reilly, juin 2014)
- *PHP Hacks*, par Jack D. Herrington (O'Reilly, décembre 2005)
- *PHP Solutions, Dynamic Web Design Made Easy, 3rd edition*, par Davis Powers (Apress, 2014)
- *CouchDB and PHP Web Development Beginner's Guide*, de Tim Juravich, (Packt Publishing, juin 2012)
- *Modern PHP, New Features and Good Practices*, par Josh Lockhart (O'Reilly, février 2015)
- *Webbots, Spiders, and Screen Scrapers, 2nd Edition (A Guide to Developing Internet Agents with PHP/CURL)*, de Michael Schrenk, NoStarch (Mars 2012)
- *Instant PHP Web Scraping*, de Jacob Ward (Packt Publishing, Juillet 2013)

Quelques ouvrages incontournables consacrés à SQL :

- *SQL Cookbook*, par Anthony Molinaro (O'Reilly)
- *SQL Hacks*, par Andrew Cumming et Gordon Russel (O'Reilly)
- *SQL Antipatterns, Avoiding the Pitfalls of Database Programming*, par Bill Karwin (PragProg)

De nouveaux ouvrages consacrés à PHP, et en particulier à PHP7, paraissent régulièrement chez les éditeurs suivants :

- Packt Publishing
- Leanpub
- APress
- Sitepoint
- PHPArchitect
- O'Reilly
- PragProg (The Pragmatic Programmer)

Il y a beaucoup de livres très intéressants parus récemment chez Leanpub, proposant beaucoup de bonnes pratiques autour de PHP. C'est un site à surveiller de près.

Pour un tour d'horizon des ouvrages en français consacrés à PHP, le plus simple c'est de passer par le site d'Eyrolles :

<http://www.eyrolles.com/Accueil/Recherche/?q=php>

En matière de bonnes pratiques relatives à PHP, une référence incontournable, c'est le site de Josh Lockhart, « PHP The right way » :

<http://www.phptherightway.com/>

... qui par chance existe aussi en français (vous trouverez le lien vers la version française sur le site officiel).

Josh Lockhart a également écrit le livre « PHP : the Right Way », en partenariat avec Phil Sturgeon, et on peut se procurer ce livre gratuitement chez Leanpub :

<https://leanpub.com/phptherightway>

Le site Startutorial propose une newsletter consacrée à PHP, et de temps à autre, le site propose des tutoriels intéressants à étudier. Exemple de tutoriel paru récemment :

https://www.startutorial.com/articles/view/php_file_upload_tutorial_part_1

Le site Developerworks d'IBM propose un grand nombre de tutoriels intéressants consacrés à PHP. J'avais déjà cité les tutoriels de Jack Herrington consacrés aux design patterns, mais vous trouverez beaucoup d'autres dossiers intéressants sur ce site :

<https://www.ibm.com/developerworks/>

De nombreux tutos intéressants disponibles sur Developpez.com :

<https://www.developpez.com/>

Le site PHPClasses.org fourmille de bonnes idées, c'est aussi un site incontournable pour les développeurs PHP :

<https://www.phpclasses.org/>

7 Changelog

Version 1.0 publiée le 18/07/2017 :

- Publication initiale (provisoire)

Version 1.1 publiée le 19/07/2017 :

- Correction de quelques coquilles
- Ajout du chapitre 5.6 (pièges du « select multiple »)
- Rédaction du chapitre 4.4.5 (générateur de formulaire)

Version 1.2 publiée le 05/08/2017 :

- Ajout à la fin du chapitre 4.2.1 d'un exemple de test basé sur l'instruction « switch ».
- Ajout de complément à la fin du chapitre 4.2.5, sur le typage des paramètres de fonctions
- Ajout de quelques conseils relatifs à Netbeans, à la fin du chapitre 2.2
- Modification de la classe GenForm dans le chapitre 4.4.5.2 (ajout possibilité de générer des attributs « disabled »)
- Création du chapitre 4.3 (dates, sessions et cookies)
- Création du chapitre 4.5 (objets)
- Création du chapitre 4.6 (PHP et SQL)
- Création du chapitre 4.8 (fichiers)
- Création du chapitre 4.9 (études de cas)
- Création des chapitres « annexes » 5.7, 5.8 et 5.9.

Version 1.3 publiée le 11/09/2017 :

- Révision du chapitre 4.9.3 (ajout du jeu de données sous forme de fichier SQL externe au support de cours, pour une réutilisation plus facile)