Développement Web 2
TP/TD - N°4

Exercice 1 – Architecture MVC

Sur la base du code présenté au cours 3, cet exercice a pour but de vous apprendre mettre en place une architecture MVC.

- Récupérez votre code du 2è exercice du TP ou la correction déposée sur le site du cours.

- Mettez en place une architecture MVC pour ce site en respectant les ajouts de fonctionnalités effectué lors du dernier TP. Aidez-vous de l'archive $premierMVC$ déposée sur le site du cours.

Exercice 2 – Contrôleur frontal

Objectifs : Factoriser le code ; utiliser des interfaces.
Source : https://www.sitepoint.com/front-controller-pattern-1/

- Modifiez le code du contrôleur frontal (fichier $index.php$ du projet $premierMVC$ vu en cours) afin de pouvoir passer des paramètres à la méthode demandée du contrôleur. Par exemple, la requête $http://example.com/index.php/courses/voir/Php$ déclenchera la méthode $view$ du contrôleur Courses (classe qui est une extension de la classe Controller) le paramètre Php.

  - Réorganisez le code du fichier $index.php$ en un introduisant une classe qui implémentera l'interface $iFronController$ suivante :

```php
<?php
interface iFrontController {
  public function setController($controller);
  public function setAction($action);
  public function setParams(array $params);
  public function run();
}
```

(Presque) toutes les autres fonctions du fichier $index.php$ pourront devenir des méthodes privés ou protégés de la classe $FrontController$. On pourra alors appeler le contrôleur frontal depuis le point d'entrée $index.php$ de cette façon :

```php
<?php
$pathToLibrary= __DIR__ . "/Library/";
require_once $pathToLibrary. "FrontController.php";
$frontController = new FrontController();
$frontController->run();
```

Autrement on pourra créer un contrôleur à la main (cela peut se révéler utile par exemple lors de tests automatisés d'une application web) de cette façon :

```php
<?php
$pathToLibrary= __DIR__ . "/Library/";
require_once $pathToLibrary. "FrontController.php";
$frontController = new FrontController();
$frontController->setController('courses');
$frontController->setActions('voir');
$frontController->setParams(['Php']);
$frontController->run();
```

Dans le projet $premierMVC$ vu en cours, nous avons utilisé la méthode $view$ de la classe Loader :

```php
public function view($view, $data = []) {
  foreach ($data as $key => $value) {
    $$key = $value;
  }
  include "views/$view.html.php";
}
```

- Nous allons ajouter à la classe $Loader$ une version modifiée de cette fonction, que nous allons nommer $render$. Au lieu d'afficher sur le tampon de sortie le contenu de la vue, cette fonction retournera la vue complétée de ses données sous la forme d'une chaîne de caractères contenant du code HTML. Conseils : utilisez les fonctions $ob\_start()$ et $ob\_get\_clean()$ de PHP, regardez pour cela la documentation de PHP.

- Modifiez la fonction $render$ de façon que toute la chaîne de caractères (récursivement) dans le tableau $data$ soit échappée (pour cela, vous pouvez utiliser $htmlentities$ ou un filtre de nettoyage).

- Ajoutez un troisième paramètre (un drapeau booléen qui sera vrai par défaut) à cette fonction, qui contrôlera si échapper ou non toutes les chaînes de caractères dans le tableau $\$data$. Échapper les chaînes sera le comportement par défaut de la fonction. Ainsi, le prototype de cette fonction sera le suivant :

```
1. public function render($view, $data = [], $escape = true) {
```

- Donnez un exemple de comment combiner plusieurs vues en un seul document HTML en utilisant plusieurs appels à cette fonction.

Exercice 3 – Autoloading

On conseille toujours de séparer les classes dans différents fichiers. Le problème c'est que l'on est obligé ensuite de faire beaucoup de $require$ pour charger nos différentes classes. Heureusement l'autoloading de PHP nous permet de remédier à ce problème en incluant les classes dès que l'on en a besoin. Chaque fois que PHP crée un nouvel objet par le mot clé new, si la définition de la classe n'est pas déjà en mémoire, PHP exécute une liste de fonctions qui vont essayer de charger la classe en mémoire. Nous pouvons ajouter nos propres fonctions dans cette liste, grâce à la fonction $spl\_register\_autoload$. Par exemple :

```
1. <?php
2. function monAutoload($className) {
3. $fileName = "$className.php";
4. $pathToClasses = 'lib/';
5. require_once "$pathToClasses/$fileName";
6. }
7. spl_autoload_register('monAutoload');
```

- Ajoutez ce code (ou un code similaire, selon la structure du projet) en première ligne du fichier $index.php$ du projet $premierMVC$ et éliminez du projet chaque utilisation des fonctions $require, require\_once, include$ pour l'inclusion d'un fichier qui contient une classe.

- Modifiez le code de la fonction $monAutoload$ afin que la recherche d'un fichier se fasse de façon récursive dans un répertoire donné. Vous pouvez utiliser des fonctions de PHP comme $file\_exists, scandir, is\_dir$.

- Quel problème peut soulever une telle recherche récursive d'une classe dans l'arborescence des fichiers ? Proposez des solutions pour résoudre ce problème.

Exercice 4 – MVC

Objectifs : *Maîtriser le jargon de MVC ; se familiariser avec le workflow MVC ; réutiliser le code exposé en cours ; distinguer entre utiliser un framework, et écrire un framework.*

Nous allons nous servir du code vu pendant le dernier cours pour mettre en œuvre une application web minimale.

- Allez sur la page du cours et, depuis le répertoire code, téléchargez l'archive $miniFramework.zip$; unzippez-le afin de créer un nouveau projet.

- Ajoutez au code du framework un fichier sql contenant la définition d'une base de données, contenant une table $Produit$ avec trois colonnes : id, nom, $description$. Peuplez le fichier sql (et donc la base) avec des produits de votre choix.

- Utilisez le $Makefile$ pour créer la base de données.

Ensuite et dans l'ordre :

- Créez une vue qui affiche tous les produits.

- Créez un modèle $Produit\_model$, avec une méthode permettant de récupérer toutes les entrées de la table $Produit$.

- Créez un contrôleur (sur le modèle du $Defaultcontroller$) qui cherche tous les produits disponibles et les affiche en utilisant la vue.

- Mettez à jour le fichier de configuration.

- Testez, déboguez, testez . . .

Pour terminer, listez (dans un fichier texte, comme par exemple un $TODO$) toutes les étapes nécessaires pour ajouter à cette application une page qui permet d'ajouter des produits dans la base de données.

Exercice 5 – Stockage des mots de passes

Objectifs : *Comprendre ce que c'est une table arc-en-ciel.*

- Choisir 10 mots, dont l'usage est (à votre avis) moins fréquent. Encodez ces mots avec l'algorithme MD5. (Vous pouvez utiliser l'outil linux $md5$, ou à défaut écrire un script PHP (réutilisez le formulaire de cryptage du TP2) qui demande un mot en entrée où en paramètre, et qui retourne son encodage MD5 : variable $\$argv$ et fonctions $readline()$, $md5$ de PHP).

- Pour chaque mot encodé, vérifiez si le mot se trouve dans la table de traduction (table arc-en-ciel) dont l'accès se trouve ici : http://hashtoolkit.com/decrypt-md5-hash/