Développement Web 2
TP/TD - N°3

Exercice 1 – Intégration d'une base de données

Sur la base du code présenté en cours, cet exercice a pour but de vous apprendre à manipuler une base de données via la création d'un site permettant de s'inscrire à un ou plusieurs cours et de visualiser les personnes inscrites.
Votre dossier de projet présentera une arborescence similaire à celle proposée ci-dessous :

```
integrationDb
    │   index.php
    │
    ├──code
    │       tools.php
    │
    ├──config
    │       pages.inc.php
    │
    ├──css
    │       bootstrap.css
    │
    ├──db
    │       create.sql
    │       database.sqlite
    │
    ├──pages
    │       registrations.php
    │       registration_create.php
    │       registration_new.php
    │
    └──templates
            error.php
            footer.html.php
            header.html.php
            menu.html.php
            registration_form.html.php
            registration_list.html.php
            success.php
```

- Récupérer les codes sources disponible sur le site du cours.

- Visualiser le site produit par ce code.

- Dans un fichier texte, identifier le rôle de chaque fichier et le rôle de toutes les fonctions présentes dans $tools.php$.

- Afin de mieux comprendre la partie SQL du code, vous pouvez vous aider d'un logiciel permettant de manipuler les bases SQLite au travers d'une interface graphique (ex : http://sqlitebrowser.org/). Ajoutez des nouveaux élèves dans la table et regardez les modifications apportées à la table.

Exercice 2 – Ajout de fonctionnalités

Nous allons modifier le site en rajoutant diverses fonctionnalités :

- Modifiez la requête appelée dans la fonction $registrations$ de tools.php afin de ne plus afficher des doublons sur la page Inscriptions lorsqu'un élève s'inscrit deux fois. (Indice : Modification de ORDER BY).

- Une fois cette modification réalisée, un élève pourrait avoir deux fois le même cours en s'inscrivant deux fois. Modifiez le code afin qu'un élève ne puisse plus se réinscrire à un cours si il est déjà inscrit à celui-ci.

- Sur la page $Inscriptions$, rajoutez un formulaire sous forme de menu déroulant avec tous les cours permettant à l'utilisateur de n'afficher que les élèves inscrit au cours sélectionné dans le menu. Plusieurs solutions sont possibles :
  - Modification de la requête SQL
  - Ajout d'une condition dans le fichier $registration\_list.html.php$

- Rajoutez dans la base de données les trois cours suivants : $HTML$, CSS, JS

- Modifiez le code pour que, dorénavant, un élève ne puisse s'inscrire au cours PHP si seulement il est déjà inscrit aux trois nouveaux cours rajoutés ci-dessus.

- Ajoutez une nouvelle fonctionnalité permettant à un élève de se désinscrire d'un ou plusieurs cours.

Exercice 3 – 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();
```

Exercice 4 – Contrôleur frontal (suite)

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 :

```php
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 5 – Contrôleur frontal (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 :

```php
<?php
function monAutoload($className) {
$fileName = "$className.php";
$pathToClasses = ’lib/’;
require_once "$pathToClasses/$fileName";
}
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.