Architecture MVC en PHP*

Partie 1 : Isolez le modèle et la vue

C'est votre première semaine dans une agence de développement, où vous venez d'être accepté en stage. Votre chef de projet vous confie d'emblée une mission. Il va falloir reprendre le blog de l'Association de Volley-Ball de Nuelly, l'AVBN. À l'époque, il avait été fait par un développeur amateur, mais maintenant que le club vient d'être promu dans la division supérieure, il leur faut un outil un peu plus robuste.

1 Préliminaires

1.1 Gestion du code avec Git et GitHub

1.1.1 Initialisation d'un dépôt local Git

Vous allez travailler avec Git. Pour ce faire, vous devez créer un dépôt local, c'est à dire un dossier dans lequel toutes vos modifications seront enregistrées.

1. Créez un dossier nommé `mvc-php` dans le répertoire `~\UwAmp\www`.
2. Placez-vous dans ce dossier `mvc-php` nouvellement créé et ouvrez une invite de commande Git Bash ou Windows PowerShell.
3. Lancer la commande `git init`. Elle initialise ce dossier `mvc-php` comme un dépôt.

Un dossier caché `.git` a été créé. Il contient tous les éléments non visibles de Git : la configuration, les logs, les branches...

4. Extraire le contenu du fichier `mvc-php.zip` directement à la racine de votre dossier `mvc-php`.
5. Passez la commande `git add -all` pour indexer le tout nouveau contenu du dépôt.
6. Utilisez la commande `git commit -m "base de travail"` pour créer une première version.

L'argument `-m` permet de définir un message particulier rattaché au commit effectué. Si vous n'utilisez pas cet argument, la commande `git commit` vous demandera de saisir le message de commit.

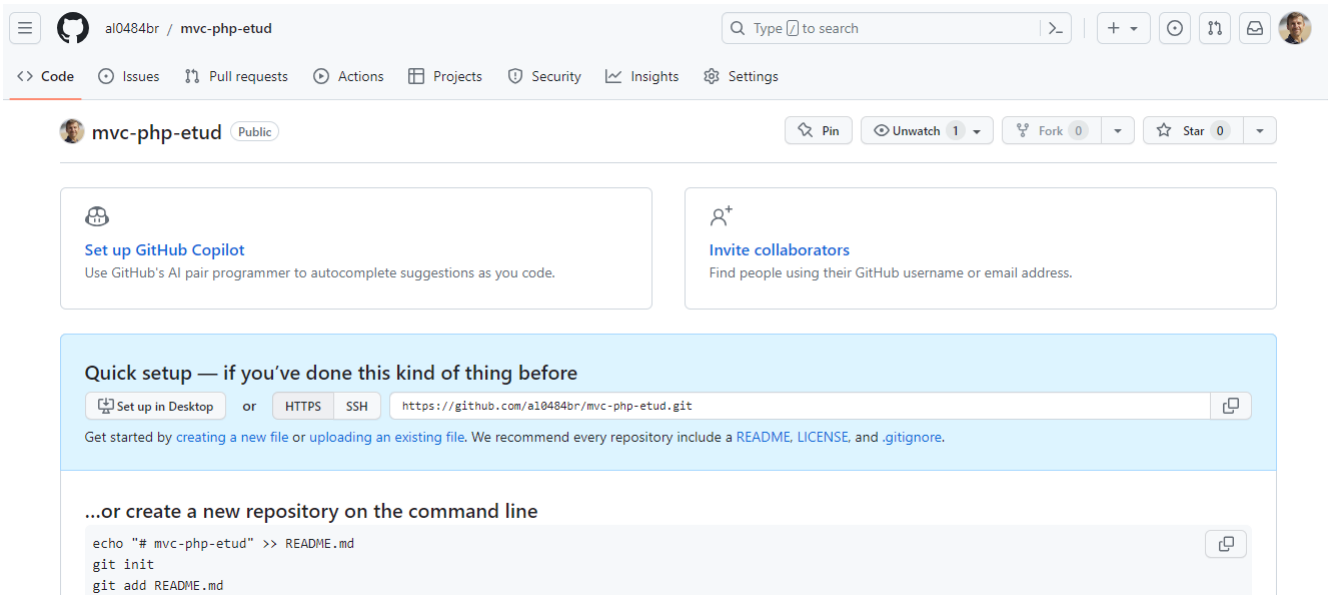
*Mathieu Nebra

1.1.2 Création d'un dépôt distant GitHub

Pour mettre votre projet sur GitHub, vous devez d'abord créer un repository (nommez-le aussi mvc-php) dans lequel il pourra être installé.

Puis vous allez devoir relier votre dépôt local au dépôt distant que vous venez de créer sur GitHub.

Pour cela, copiez le lien https (de la forme `https://github.com/al0484br/mvc-php-etud.git`) qui se trouve en haut sur fond bleu dans l'onglet Code (voir capture d'écran ci-après) et collez-le dans la commande `git remote add origin https://github.com/al0484br/mvc-php-etud.git`.



Ensuite tapez la commande `git branch -M main`.

Vous avez relié le dépôt local au dépôt distant. Vous pouvez donc envoyer des commits du repository vers le dépôt distant GitHub en utilisant la commande `git push -u origin main`

Au fait, n'oubliez pas de m'inviter en tant que collaborateur (alexbrabant) ;-)

1.2 Création de la Base de données

Par défaut, l'application utilise une base de données MySQL dénommée `blog`, accessible à un utilisateur `blog` dont le mot de passe est `password`.

Vous devez remplacer les identifiants utilisés dans le code par les vôtres dans le fichier `blog/index.php`, à la ligne 17 :

```
$bdd = new PDO('mysql:host=localhost;dbname=blog;charset=utf8', 'blog', 'password');
```

Vous devez aussi remplir votre base de données. Vous pouvez charger le schéma par défaut (et quelques données), contenu dans le fichier `db.sql`.

1.3 Lancement

Vous pouvez ouvrir votre navigateur préféré et charger la page `http://localhost/mvc-php/blog/`.

1.4 Défauts du code existant

Le code de la page d'accueil dont une capture d'écran est donnée ci-avant se trouve dans le fichier `index.php`.

Quels sont les défauts de ce code ? Citez-en au moins 3 importants avec exemples à l'appui.

2 Isolez l'affichage du traitement PHP

Une bonne pratique consiste à éviter de mélanger l'affichage du reste, c'est à dire de séparer le code HTML du code PHP.
Vous allez refactoriser le code existant, sans ajouter de nouvelles fonctionnalités. Le but est d'avoir un code plus facile à modifier par la suite.

2.1 Première séparation

Vous allez séparer le code en deux sections :

1. On récupère les données. C'est là qu'on fait notamment la requête SQL. Vous voyez aussi qu'on crée une variable structurée, appelée `$posts`, qui contient les "données brutes" utiles à l'affichage de chaque billet de blog.
2. On affiche les données en les formatant comme on le souhaite avec les fonctions `nl2br()` et `htmlspecialchars()`, par exemple.

Le screencast `1re_separation.asf` détaille chacune de ces modifications pas à pas.

2.2 Séparation du code en 2 fichiers

Conservez le fichier `index.php` (il y a toujours un `index.php`, c'est le fichier de base de votre site, il est lu en premier) : mais ce fichier ne s'occupera plus que de récupérer les données et d'appeler l'affichage.
Créez un répertoire `templates` et créez dans celui-ci un fichier `homepage.php` qui s'occupera d'afficher les données dans la page.

Vous avez créé un dossier qui contiendra seulement les gabarits d'affichage (les templates) de chacune des pages du blog. Les templates sont séparés du code responsable d'agir sur les données. Vous verrez plus tard dans votre apprentissage sur les frameworks (Symfony ou Laravel, par exemple), que c'est une pratique courante qui rend le code plus maintenable et favorise le travail en équipe.

Déplacez le code HTML dans le fichier `templates/homepage.php` de manière à ce que le fichier `index.php` ne contienne plus que le code PHP :

`templates/homepage.php` :

```
<!DOCTYPE html>
<html>
        <head>
            <meta charset="utf-8" />
            <title>Le blog de l'AVBN</title>
            <link href="style.css" rel="stylesheet" />
        </head>

        <body>
            <h1>Le super blog de l'AVBN !</h1>
            <p>Derniers billets du blog :</p>

            <?php
            foreach ($posts as $post) {
            ?>
                <div class="news">
                    <h3>
                        <?php echo htmlspecialchars($post['title']); ?>
                        <em>le <?php echo $post['french_creation_date']; ?></em>
                    </h3>
                    <p>
                        <?php
```

```
                        // We display the post content.
                        echo nl2br(htmlspecialchars($post['content']));
                        ?>
                        <br />
                        <em><a href="#">Commentaires</a></em>
                    </p>
                </div>
            <?php
            } // The end of the posts loop.
            ?>
        </body>
</html>
```

Ensuite, ajoutez l'instruction PHP `require('templates/homepage.php');` à la fin du fichier `index.php` pour appeler `templates/homepage.php` :

`index.php` :

```
<?php

// We connect to the database.
try {
        $database = new PDO('mysql:host=localhost;dbname=blog;charset=utf8',
        'blog', 'password');
} catch(Exception $e) {
        die('Erreur : '.$e->getMessage());
}

// We retrieve the 5 last blog posts.
$statement = $database->query(
        "SELECT id, titre, contenu, DATE_FORMAT(date_creation,
        '%d/%m/%Y à %Hh%imin%ss') AS date_creation_fr
        FROM billets ORDER BY date_creation DESC LIMIT 0, 5"
);
$posts = [];
while (($row = $statement->fetch())) {
        $post = [
            'title' => $row['titre'],
            'french_creation_date' => $row['date_creation_fr'],
            'content' => $row['contenu'],
        ];

        $posts[] = $post;
}

require('templates/homepage.php');
?>
```

Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all
> git commit -m "Isolez l'affichage du traitement PHP"
> git push -u origin main
```

3 Isolez l'accès aux données

Nous allons maintenant séparer complètement l'accès aux données (tout le traitement SQL) dans un fichier spécifique.

Nous aurons 3 fichiers :

- `src/model.php` : se connecte à la base de données et récupère les billets.
- `templates/homepage.php` : affiche la page. Ce fichier ne va pas changer du tout.
- `index.php` : ne se contente plus que de faire le lien entre le modèle et l'affichage.

On y reviendra, mais sachez que ces 3 fichiers forment la base d'une structure MVC :

- Le modèle traite les données (`src/model.php`).
- La vue affiche les informations (`templates/homepage.php`).
- Le contrôleur fait le lien entre les deux (`index.php`).

3.1 Le modèle

Notre nouveau fichier `src/model.php` contient une fonction `getPosts()` qui renvoie la liste des billets.

`src/model.php` :

```php
<?php

function getPosts() {
        // We connect to the database.
        try {
            $database = new PDO('mysql:host=localhost;dbname=blog;charset=utf8',
            'blog', 'password');
        } catch(Exception $e) {
            die('Erreur : '.$e->getMessage());
        }

        // We retrieve the 5 last blog posts.
        $statement = $database->query(
            "SELECT id, titre, contenu, DATE_FORMAT(date_creation,
```

```
            '%d/%m/%Y à %Hh%imin%ss') AS date_creation_fr FROM billets
            ORDER BY date_creation DESC LIMIT 0, 5"
        );
        $posts = [];
        while (($row = $statement->fetch())) {
            $post = [
                'title' => $row['titre'],
                'french_creation_date' => $row['date_creation_fr'],
                'content' => $row['contenu'],
            ];

            $posts[] = $post;
        }

        return $posts;
}

?>
```

3.2 La vue

Comme on l'a dit, l'affichage n'a pas du tout changé donc on ne touche pas à `templates/homepage.php`.

3.3 Le contrôleur

La nouveauté, c'est désormais l'index (aussi appelé le contrôleur) qui sert d'intermédiaire entre le modèle et la vue. Ce fichier ne contient plus que 3 lignes :

1. On charge le fichier du modèle. Il ne se passe rien pour l'instant, parce qu'il ne contient que la définition de la fonction `getPosts()`.
2. On appelle la fonction `getPosts()`, qui renvoie la liste des billets dans une variable locale que l'on nomme `$posts`.
3. On charge le fichier `templates/homepage.php`, qui va présenter les informations dans une page HTML.

`index.php` :

```
<?php

require('src/model.php');

$posts = getPosts();

require('templates/homepage.php');
?>
```

Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all
> git commit -m "Isolez l'accès aux données"
> git push -u origin main
```

4 Améliorations esthétiques

Votre code est maintenant découpé en 3 fichiers :

- Un pour le traitement PHP : il récupère les données de la base ☞ le modèle.
- Un pour l'affichage : il affiche les informations dans une page HTML ☞ la vue.
- Un pour faire le lien entre les deux : ☞ le contrôleur.

Cette structure est bonne, mais nous allons faire quelques améliorations cosmétiques. Ce ne sont peut-être que des détails pour vous, mais ça fera une vraie différence à la fin.

Voici ce que nous allons améliorer :

- l'anglais ;
- la balise de fermeture PHP ;
- l'utilisation de short open tags.

4.1 Le contrôleur

Enlever la balise de fermeture `?>` à la fin du fichier `index.php`.

Les développeurs professionnels enlèvent la balise de fermeture PHP dans les pages qui ne contiennent que du PHP. Cela permet d'éviter que ces fichiers n'envoient par erreur du code HTML sous forme d'espaces blancs alors qu'ils ne devraient pas. C'est d'ailleurs une recommandation officielle de la norme de formattage PSR-12 qui dit :

« The closing `?>` tag MUST be omitted from files containing only PHP. »

4.2 Le modèle

Commencez par enlever la balise de fermeture `?>` à la fin du fichier `src/model.php` .

Pour avoir du code uniquement en anglais, vous devez renommer la base de données en anglais. Pour l'instant, nous avons une seule table `posts` (au lieu de billets) qui contient 4 champs, renommés en anglais ci-après :

- `id` (integer)
- `title` (varchar)
- `content` (text)
- `creation_date` (datetime)

N'ayez pas peur d'utiliser de l'anglais, même si vous le parlez mal. En programmation, écrire en mauvais anglais est préférable à écrire en bon français (!).

4.3 La vue

Ce n'est pas obligatoire, mais profitons en pour découvrir les short echo tags. C'est un raccourci en PHP pour faciliter la lisibilité du code. Ainsi, par exemple :

```php
<?php echo htmlspecialchars($post['title']); ?>
```

devient :

```php
<?= htmlspecialchars($post['title']) ?>
```

Cela permet d'éviter d'avoir à écrire `echo` quand on souhaite juste afficher une variable.
Le but est d'être plus lisible dans la vue en enlevant le maximum de code PHP là-dedans (même si on ne peut pas tout enlever).

Les frameworks PHP tels que Symfony ont carrément développé un langage de template (appelé Twig pour Symfony), qui permet de ne pas utiliser du tout de code PHP dans notre affichage.

Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all
> git commit -m "Améliorations esthétiques"
> git push -u origin main
```