Architecture MVC en PHP*

Partie 2 : Factorisez votre code dans une architecture MVC

Chapitre 2 : Affichez des commentaires

1 Préparation de l'environnement de travail

1.1 Dépôt Git

Vous allez continuer à travailler avec votre dépôt local Git créé lors de la séance précédente dans le répertoire `~\UwAmp\www\php-mvc`. Pour ce faire :

1. Placez-vous donc dans le dossier `mvc-php` et ouvrez une invite de commande Git Bash ou Windows PowerShell.
2. Décompressez le contenu du fichier `mvc-php-affichez-commentaires.zip` directement à la racine de votre dossier `mvc-php` en acceptant de remplacer les fichiers existant avec ces nouveaux fichiers.

1.2 Base de données

Nous réutilisons la base de données MySQL dénommée `blog`, créée lors de la précédente séance.

Vous devrez éventuellement remplacer les identifiants de connexion à la base de données utilisés dans le code par les vôtres dans le fichier `src/model.php`, à la ligne 5 :

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

Votre base de données est en principe déjà remplie. Vous pouvez recharger le schéma par défaut (et quelques données), contenu dans le fichier `db.sql`.

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

```
> git add --all
> git commit -m "Base pour afficher les commentaires"
> git push -u origin main
```

*Mathieu Nebra

2 Affichez des commentaires

Ça y est, on vient de vous demander une nouvelle fonctionnalité ! On aimerait pouvoir afficher une page avec les commentaires de chaque billet.

Vous vous souvenez du lien "Commentaires" sous chaque billet ?

Il est temps de faire marcher ce lien "Commentaires" !

Lorsqu'on clique dessus, on va afficher une page avec le billet et sa liste de commentaires.

Hum, mais comment s'y prendre avec une architecture MVC ?

Je vous propose le plan suivant pour arriver à vos fins :

- Vous commencez par écrire la **vue**. Après tout, votre objectif principal reste d'afficher la page des commentaires à l'utilisateur !
- Ensuite, vous allez écrire un **contrôleur**, mais en version très rapide, qui fera passer des fausses données à la vue. Ça vous permettra de vérifier que votre affichage correspond à vos attentes.
- Vous affinerez le **contrôleur** en le rendant dynamique et en commençant à imaginer les services que vous souhaiteriez demander à votre modèle.
- Vous finirez en implémentant votre **modèle**, pour qu'il réponde correctement aux demandes de votre contrôleur.

Ne vous inquiétez pas, je vais vous guider pour mettre en oeuvre ce plan de travail.

Vous êtes prêts, allez, c'est parti !

2.1 Créez la vue

Qui dit nouvelle vue, dit nouveau fichier dans notre dossier `templates/`. On veut afficher un article de blog, alors nous pouvons l'appeler `templates/post.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><a href="index.php">Retour à la liste des billets</a></p>

        <div class="news">
            <h3>
                <?= htmlspecialchars($post['title']) ?>
                <em>le <?= $post['french_creation_date'] ?></em>
            </h3>

            <p>
                <?= nl2br(htmlspecialchars($post['content'])) ?>
            </p>
        </div>

        <h2>Commentaires</h2>

        <?php
        foreach ($comments as $comment) {
        ?>
        <p><strong><?= htmlspecialchars($comment['author']) ?></strong>
         le <?= $comment['french_creation_date'] ?></p>
        <p><?= nl2br(htmlspecialchars($comment['comment'])) ?></p>
        <?php
        }
        ?>
    </body>
</html>
```

Dans cette vue, on affiche :

- le billet : on utilise une variable tableau `$post`, avec les `indextitle`, `french_creation_date` et `content` ;
- les commentaires : un tableau de tableaux, avec les index `author`, `french_creation_date` et `comment`.

Pour l'instant, on ne peut pas encore tester notre code HTML. Il nous faut un contrôleur.

2.2 Créez le contrôleur "simplet"

Nous avions déjà écrit un contrôleur `index.php` pour gérer la liste des derniers billets. Je vous propose d'écrire un autre contrôleur `post.php` qui affiche un post et ses commentaires.

```php
<?php

$post = [
    'title' => 'Un faux titre.',
    'french_creation_date' => '03/03/2022 à 12h14min42s',
    'content' => "Le faux contenu de mon billet.\nC'est fantastique !",
];
$comments = [
    [
        'author' => 'Un premier faux auteur',
        'french_creation_date' => '03/03/2022 à 12h15min42s',
        'comment' => 'Un faux commentaire.\n Le premier.',
    ],
    [
        'author' => 'Un second faux auteur',
        'french_creation_date' => '03/03/2022 à 12h16min42s',
        'comment' => 'Un faux commentaire.\n Le second.',
    ],
];

require('templates/post.php');
```

Vous remarquez que mon contrôleur n'inclut pas le modèle. Je vous l'avais dit : dans un premier temps, je cherche seulement à **tester que mon affichage correspond à mes attentes**. Vous n'avez pas encore besoin de données dynamiques à ce stade et ça ne vous coûte presque pas de temps de créer ces variables manuellement. Maintenant, si vous allez sur la page `/post.php` dans votre navigateur, vous pouvez vérifier : votre page de billet affiche bien l'article, suivi des commentaires.

Cette page fonctionne, même si elle contient de fausses données !

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

```
> git add --all
> git commit -m "Créez la vue et le contrôleur"
> git push -u origin main
```

2.3 Dynamisez le contrôleur

Pour commencer, on veut que le contrôleur **prenne en paramètre un billet précis**. Il va falloir modifier les liens présents sur la page d'accueil `templates/homepage.php`, à la ligne 24, pour y renseigner notre nouvelle URL. On choisit d'utiliser le paramètre GET intitulé `id` pour faire passer l'information à notre nouveau contrôleur.

```
<!-- templates/homepage.php:24 -->

<em>
<a href="post.php?id=<?= urlencode($post['identifier']) ?>">Commentaires</a>
</em>
```

On a besoin d'une nouvelle propriété `identifier` au niveau de chaque `$post` de notre page d'accueil. Et on va directement demander à notre modèle de nous la donner :

```php
<?php
// src/model.php:15

$post = [
    'title' => $row['title'],
    'french_creation_date' => $row['french_creation_date'],
    'content' => $row['content'],
    'identifier' => $row['id'],
];
```

Vérifiez que lorsque vous cliquez sur les liens "Commentaires" de la page d'accueil, vous accédez dorénavant à votre nouvelle page.

Modifions maintenant le contrôleur `post.php` pour prendre en compte ce paramètre GET :

```php
<?php
// post.php

require('src/model.php');

if (isset($_GET['id']) && $_GET['id'] > 0) {
    $identifier = $_GET['id'];
} else {
    echo 'Erreur : aucun identifiant de billet envoyé';

    die;
}

$post = getPost($identifier);
$comments = getComments($identifier);

require('templates/post.php');
```

Il vérifie qu'on a bien reçu en paramètre un id dans l'URL (`$_GET['id']`).

Ensuite, il appelle les 2 fonctions du modèle dont on va avoir besoin : `getPost` et `getComment`. On récupère ça dans nos deux variables. Bien sûr, on attend du modèle qu'il nous renvoie les mêmes propriétés que celles qu'on avait définies :

- `title`, `french_creation_date` et `content` pour les billets ;
- `author`, `french_creation_date` et `comment` pour les commentaires.

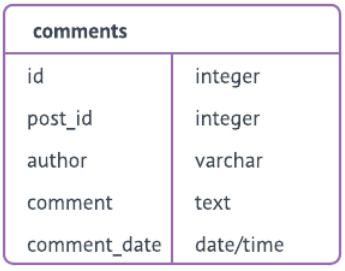
Testez votre code et constatez qu'il ne fonctionne plus (message « Erreur : aucun identifiant de billet envoyé »). C'est normal puisqu'on a demandé de l'aide au modèle, mais on n'a pas encore implémenté ces fonctions.

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

```
> git add --all
> git commit -m "Dynamisez le contrôleur"
> git push -u origin main
```

2.4 Mettez à jour le modèle

Tout d'abord, ajoutons (si ce n'est pas déjà fait) une table pour gérer les commentaires dans la base. Elle aura cette structure :



La structure de la table comments

Pour avoir directement tout à jour, vous pouvez charger le fichier `db.sql` fourni.

Maintenant, concentrons-nous sur le code du modèle. Pour l'instant, notre fichier `src/model.php` ne contient qu'une seule fonction `getPosts` qui récupère tous les derniers posts de blog.

On va écrire 2 nouvelles fonctions :

- `getPost` (au singulier !), qui récupère un post précis en fonction de son ID ;
- `getComments`, qui récupère les commentaires associés à un ID de post.

Cela donne :

```php
<?php
// src/model.php
```

```php
function getPosts()
{
    // ... (déjà écrite)
}

function getPost($identifier) {
    try {
        $database = new PDO('mysql:host=localhost;dbname=blog;charset=utf8',
        'root', 'root');
    } catch(Exception $e) {
        die('Erreur : '.$e->getMessage());
    }

    $statement = $database->prepare(
        "SELECT id, title, content,
        DATE_FORMAT(creation_date, '%d/%m/%Y à %Hh%imin%ss')
        AS french_creation_date FROM posts WHERE id = ?"
    );
    $statement->execute([$identifier]);

    $row = $statement->fetch();
    $post = [
        'title' => $row['title'],
        'french_creation_date' => $row['french_creation_date'],
        'content' => $row['content'],
    ];

    return $post;
}

function getComments($identifier)
{
    try {
        $database = new PDO('mysql:host=localhost;dbname=blog;charset=utf8',
        'root', 'root');
    } catch(Exception $e) {
        die('Erreur : '.$e->getMessage());
    }

    $statement = $database->prepare(
        "SELECT id, author, comment,
        DATE_FORMAT(comment_date, '%d/%m/%Y à %Hh%imin%ss')
        AS french_creation_date FROM comments WHERE post_id = ?
        ORDER BY comment_date DESC"
    );
    $statement->execute([$identifier]);
```

```
    $comments = [];
    while (($row = $statement->fetch())) {
        $comment = [
            'author' => $row['author'],
            'french_creation_date' => $row['french_creation_date'],
            'comment' => $row['comment'],
        ];
        $comments[] = $comment;
    }

    return $comments;
}
```

Ces deux nouvelles fonctions prennent un paramètre : l'identifiant du billet qu'on recherche. Cela nous permet notamment de ne sélectionner **que** les commentaires liés au post concerné.

Hum, mais il y a quelque chose qui me dérange ici.

Je n'aime pas voir du code se répéter. C'est le cas en particulier de la connexion à la base de données avec les blocs try/catch. Allez, on va factoriser ça !

A votre tour

Je vous demande tout simplement d'ajouter une fonction `dbConnect()` qui va renvoyer la connexion à la base de données.

C'est mieux mais c'est encore améliorable car actuellement on fait une connexion à la base de données pour chaque nouvelle requête. Nous l'améliorerons plus tard.

Testez, ça doit fonctionner.

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

```
> git add --all
> git commit -m "Mettez à jour le modèle"
> git push -u origin main
```