

Part 01 - Mise en place



Votre mission sera de développer une application Web PHP pour gérer une collection de personnages d'un jeu Mihoyo.

Les 3 jeux phares ayant une structure similaires pour leurs personnages, vous êtes libre de choisir le jeu pour représenter le thème de votre TP.

Pour ajouter de la structure au projet, nous allons travailler avec un design pattern : Le MVC (Model-View-Controller). [Voir détails](#)

Votre code devra être mettre en place un suivis de version git et être hébergé sur Github (ou concurrent)

Voici nos objectifs pour tout le projet :

- ☐ Afficher la liste des personnages
- ☐ Ajouter des personnages à la BD
- ☐ Editer un personnage
- ☐ Supprimer un personnage
- ☐ Gérer une authentification
- ☐ Gérer un journal de log
- ☐ Gérer les données liés à un personnage
- ☐ Avoir un design simple et fonctionnel
- ☐ Gérer une collection de personnage

1 - Mise en place du projet

Dans votre dossier de travail (Bureau, Dossier XAMP, ...) vous allez créer un premier fichier *index.php*.

Il servira de point d'entrée de votre application. Nous allons ensuite créer quelques dossiers.

```

TonSuperProjet
├── Config
├── Controllers
├── Exceptions
├── Helpers
├── Models
├── public
│   ├── css
│   └── img
├── Services
├── Vendor
├── Views
└── index.php
```

Dans le dossier `Helpers`, créez une classe PHP nommée `Psr4AutoloaderClass.php`. Voici son contenu avec les commentaires.

[Voir code sur github](#)

N'oubliez pas de commit puis push !!

Les dossiers vides peuvent ne pas apparaître sur github. Cela est normal

2 - Chargeeeeeeeeeez !

2.1 : Dans notre `index.php`, il est temps d'utiliser notre loader. Importez le (`require_once`) puis créez une instance de `Helpers\Psr4AutoloaderClass`

2.2 : Avec votre objet loader, utilisez la fonction `register()` une fois. Cela permet de faire comprendre à PHP que c'est cette classe qui va gérer l'autoload.

2.3 : On va enregistrer le namespace `Helpers` qu'utilise notre classe. Voici la syntaxe pour que vous puissiez enregistrer les futurs namespaces de l'appli :

```
$loader->addNamespace('\Helpers', '/Helpers');
```

3 - Gérer la partie V du MVC

3.1 : Dans cette version, nous allons utiliser un moteur de template pour gérer nos vue. En php, plusieurs nom existent : Blade, Twig, ...

Dans un soucis de simplicité, nous allons utiliser Plate. [Téléchargez la version 3.5.](#)

Puis placez le dossier Plates dans le dossier `Vendor`

3.2 : Retournez dans votre fichier index.php, puis ajoutez le namespace `\League\Plates` de votre dossier `/Vendor/Plates/src`.

Normalement, toujours aucune erreur ne devrait s'afficher.

3.3 : Nous allons créer notre première vue. Dans le dossier `Views`, créez un fichier `home.php` et `template.php`.

Dans le fichier `home.php`, ajoutez ce simple code :

```
<?php
    $this->layout('template', ['title' => 'TP Mihoyo']);
?>
<h1>Collection <?= $this->e($gameName) ?></h1>
```

La première ligne permettra d'utiliser notre template de base (celui qui se répétera sur toutes nos pages). La seconde affichera un titre avec une variable

Vous pouvez notifier les balises `<?= =>` Celles ci correspondent à un raccourci PHP pour dire `<?php echo $var ?>`

3.3 : Le fichier `template.php` sert à représenter tout ce qui est présent en permanence sur notre page (menu, pied de page, logo, ...).

C'est celui-ci qui chargerait notre `css`, `js` et autres dépendances dans la balise `head`. Celui-ci aura accès à 2 variables :

- `$title` : qui contient la valeur pour la balise `title` ;
- `$content` : qui contient tout le code de notre page.


Je vais vous proposer un squelette pour votre gabarit. Il sera à compléter avec votre structure (menu par exemple), mais aussi avec les variables pour placer le contenu ou vous le désirez.

Vous pouvez faire votre propre gabarit si besoin. Notez bien la syntaxe entre afficher une variable et définir une section.

```
<!doctype html>
<html lang="fr">
```

```
<head>
  <meta charset="UTF-8"/>
  <link rel="stylesheet" href="public/css/main.css"/>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title><?= $this->e($title) ?></title>
</head>
<body>
<header>
  <!-- Menu -->
  <nav>
  </nav>
</header>
<!-- #contenu -->
<main id="contenu">
<?=$this->section('content')?>
</main>
<footer>

</footer>
</body>
</html>
```

Si vous êtes observateur, vous remarquerez une référence à un fichier CSS. Je vous recommande de le créer pour styliser votre page ( et oui le css compte dans la note).

Vous pourrez aussi voir comment afficher le contenu d'une variable vu que le titre est affiché dans la balise *title*.

3.4 : Si vous voulez tester, vous pouvez faire générer votre vu dans votre `index.php` .

Pour ce faire, instanciez un `Engine` avec le dossier où se situe vos fichiers `home` et `template`. Puis faites appel à la fonction `render()` .

Celle-ci prend 2 paramètres :

- le nom du fichier à render
- un array clé/valeur de variable à passer à la vue (la clé étant le nom passé à la fonction e dans votre template)

N'oubliez pas d'echo le retour de votre render

Si tout s'est bien déroulé, vous devriez voir votre page affichant votre *h1* et le titre dans l'onglet.

4 - Gérer le contrôleur pour afficher la vue

Il est grand temps d'afficher quelque chose ! Mais pour cela, il nous faudra un chef d'orchestre ! Le contrôleur à la rescousse.

4.1 : Nous allons créer fichier et une classe `MainController` dans le dossier `Controllers`

Pour le moment, il ne fera pas grand-chose d'autre que construire notre vue. N'oubliez pas de déclarer le namespace `Controllers`.

Il vous faudra un attribut qui stockera l'instance de l'engine initialisé dans le constructeur.

4.2 : Ajoutons une fonction `Index` qui aura pour but de générer notre vue.

```
public function index() : void {  
    echo $this->templates->render('home', ['gameName' => 'Genshin Impact']);  
}
```

Prenez bien le temps de comprendre ce que fait cette fonction. Et surtout que les paramètres ne sont pas choisis au hasard ;)

4.3 : Pour finaliser notre contrôleur, nous devons nous reposer sur un autre composant (souvent dans l'ombre) => Le "*routeur*".

Cette fois, pas besoin de créer un fichier, nous allons utiliser notre `index.php`.

Pour tester que tout fonctionne, il nous suffit d'instancier un `MainController` et d'en appeler sa méthode `Index()`.

(⚠ Chargez le namespace ⚠)

Si tout va bien, votre page devrait s'afficher avec notre balise `h1` !

Si vous aviez, au préalable, affiché votre vue dans la question précédente, n'oubliez d'enlever le code, sinon votre vue sera render 2 fois

5 - Fin de la partie 1

À la fin, ton arborescence devrait ressembler à cela :

```
TonSuperProjet  
├── Config  
├── Controllers  
│   └── MainController.php  
├── Exceptions  
└── Helpers
```

```
| └─ 📄 Psr4AutoloaderClass.php
|   └─ 📁 Models
|   └─ 📁 public
|     └─ 📁 css
|       └─ 📄 main.css
|     └─ 📁 img
|   └─ 📁 Services
|   └─ 📁 Vendor
|     └─ 📁 Plates
|   └─ 📁 Views
|     └─ 📄 template.php
|     └─ 📄 home.php
└─ 📄 index.php
```