Mini projet 2 - Création d’un Framework POO MVC

### Formateur

* PERRIGAULT Alexandre

### Objectifs (compétences)

* Compréhension de l’architecture Model / View / Controller
* Compréhension des concepts POO en PHP
* Sécuriser l’accès à son application via l’authentification

### Modalités

* **Travail en autonomie** jusqu’à l’étape 8
* **Travail en îlot** avec partage des tâches après l’étape 8

### Livrable

* Un schéma papier décrivant l’architecture MVC pour afficher une page intégrant des données dynamiques.
* Un commit GIT à la fin de chaque étape
* Le framework fonctionnel sur votre compte github

## 

## Étape 1 : Création d’un dépôt GIT

La première étape est de créer un répertoire, par exemple **myframework** qui contiendra notre projet et d’initialiser un projet GIT en son sein. Pour rappel voici la commande qui permet d’initialiser un projet GIT.

|  |
| --- |
| git init |

Nous allons créer un fichier .gitignore afin d'exclure toutes les ressources non désirées de notre historique GIT. En vous aidant du site <https://www.gitignore.io/> et en fonction de votre IDE crée le fichier .gitignore.

Nous allons travailler sur une autre branche que la branche master sur votre environnement : **[votre-prénom]-dev**. Pousser votre projet sur votre compte github. Pensez à bien envoyer la branche master et la branche votre-prénom-dev sur votre compte github et à bien travailler sur la branche votre-prénom-dev.

À chaque fin d’étape, commiter vos modifications sur votre compte github.

Voici des ressources sur l’utilisation de GIT :

* <https://www.grafikart.fr/formations/git>
* <https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html>
* <http://rogerdudler.github.io/git-guide/index.fr.html>

Voici à quoi devrait ressembler votre dépôt à la fin de cette étape : <https://github.com/le-campus-numerique/myframework/tree/0.0.1>

## Étape 2 : Initialisation du projet

On initialise notre projet avec composer, voici la commande.

|  |
| --- |
| composer init |

|  |
| --- |
| Welcome **to** the Composer config generator      This command will guide you through creating your composer.json config.  Package name (<vendor>/<name>) [aerue/myframework]:  Description []: Créer mon premier framework en PHP POO MVC Author [Alexandre PERRIGAULT <contact@alexandre-perrigault.fr>, n **to** skip]:  Minimum Stability []: dev Package Type (e.g. library, project, metapackage, composer-plugin) []:  License []: MIT  Define your dependencies.  Would you like **to** define your dependencies (require) interactively [**yes**]? **no** Would you like **to** define your dev dependencies (require-dev) interactively [**yes**]? **no**  {  "name": "aerue/myframework",  "description": "Créer mon premier framework en PHP POO MVC",  "license": "MIT",  "authors": [  {  "name": "Alexandre PERRIGAULT",  "email": "contact@alexandre-perrigault.fr"  }  ],  "minimum-stability": "dev",  "require": {} }  **Do** you confirm generation [**yes**]? |

Nous allons créer au sein de notre répertoire de projet plusieurs sous-dossiers :

* **core** qui contiendra l’ensemble des classes du coeur de notre framework
* **src** qui contiendra la logique métier de notre site
  + **Controller** qui contiendra nos contrôleurs
  + **Model** qui contiendra nos modèles
  + **View** qui contiendra nos vues
* **public** qui contiendra les fichiers statiques de notre site (.css, .js, .jpg, ect...) ainsi que le point d’entrée (index.php)

Mettons en place un autoloader avec composer afin de nous simplifier la vie.

|  |
| --- |
| ...  "autoload": {  "psr-4": {  "Core\\": "core",  "App\\": "src"  }  } |

Ce qui nous intéresse ici c'est la nouvelle entrée dans notre objet json de notre fichier **composer.json** : autoload.

Pour respecter les derniers standards de PHP, nous utiliserons le [psr-4](https://www.php-fig.org/psr/psr-4/). La déclaration de nos espaces de noms est assez simple, on définit dans un premier temps un préfixe App puis on lui affecte comme valeur le nom du dossier src.

Pour vous donner un exemple, si on crée un nouveau contrôleur, voici à quoi ressemblera le namespace :

|  |
| --- |
| <?php  namespace App/Controller;  **class** **HomeController** **extends** **AppController** {   } |

Il ne faut pas oublier de mettre à jour composer avec la commande composer update. Il va générer un ensemble de fichier pour gérer l'autoload.

|  |
| --- |
| composer update Loading composer repositories **with** package information Updating dependencies (including require-dev) Nothing to install or update Generating autoload files |

Vous avez sans doute remarqué qu'un nouveau dossier a vu le jour vendor. Ce dossier contient les fichiers nécessaires à la gestion de l'autoload par composer, mais ce dossier contiendra aussi nos dépendances externes comme blade par exemple.  
  
**N'oubliez pas d'ajouter le dossier vendor dans le fichier .gitignore si vous ne l’avez pas fait lors du composer init.**  
  
*Je vous invite à regarder les fichiers générés par composer pour comprendre un peu mieux le fonctionnement de son autoload.*

Voici à quoi devrait ressembler votre dépôt à la fin de cette étape : <https://github.com/le-campus-numerique/myframework/tree/0.0.2>

## Étape 3 : Classe Request

On va enfin écrire nos premières lignes de code, nous allons créer un fichier **index.php** dans le dossier public, ce sera la porte d’entrée de notre Framework.

Comme vous êtes censé le savoir, lorsqu'un utilisateur va sur une page web, une requête est envoyée au serveur web, ce dernier effectue un traitement puis renvoie une réponse en HTML.

Nous allons donc commencer par implémenter la classe Request. Cette classe nous permettra de gérer facilement les [superglobales de PHP](http://php.net/manual/fr/language.variables.superglobals.php). Cela nous permettra de récupérer toutes les informations de la requête envoyée par l'utilisateur.

Créer dans le core du framework, la classe Request qui devra dans son constructeur gérer les superglobales :

* $\_POST
* $\_GET
* $\_FILES
* $\_REQUEST
* $\_SERVER
* $\_COOKIE
* $\_SESSION

Créer une méthode statique createFromGlobals dans la classe Request afin de renvoyer une instance de notre classe Request.  
  
Modifier le fichier **public/index.php** pour utiliser notre classe Request.

Vous pouvez lancer un server php afin de vérifier le bon fonctionnement de la classe avec un [var\_dump()](http://php.net/manual/fr/function.var-dump.php).

|  |
| --- |
| **php** **-S** **localhost**:8000 |

Si tout se passe bien, voici ce qui devrait s’afficher sur votre écran en allant à l’url : [http://localhost:8000](http://localhost:8000/)

|  |
| --- |
| /var/www/myframework/public/index.php:9: object(Core\Request)[3]  **private** 'post' =>   array (size=0)  **empty**  **private** 'get' =>   array (size=0)  **empty**  **private** 'files' =>   array (size=0)  **empty**  **private** 'cookie' =>   array (size=1)  'PHPSESSID' => **string** '5es0t2nb7flsrjd6mmjpn3a6am' (length=26)  **private** 'session' =>   array (size=0)  **empty**  **private** 'request' =>   array (size=0)  **empty**  **private** 'server' =>   array (size=23)  'DOCUMENT\_ROOT' => **string** '/var/www/myframework' (length=20)  'REMOTE\_ADDR' => **string** '127.0.0.1' (length=9)  'REMOTE\_PORT' => **string** '36722' (length=5)  'SERVER\_SOFTWARE' => **string** 'PHP 7.2.15-1+ubuntu18.04.1+deb.sury.org+1 Development Server' (length=60)  'SERVER\_PROTOCOL' => **string** 'HTTP/1.1' (length=8)  'SERVER\_NAME' => **string** 'localhost' (length=9)  'SERVER\_PORT' => **string** '8000' (length=4)  'REQUEST\_URI' => **string** '/public' (length=7)  'REQUEST\_METHOD' => **string** 'GET' (length=3)  'SCRIPT\_NAME' => **string** '/public/index.php' (length=17)  'SCRIPT\_FILENAME' => **string** '/var/www/myframework/public/index.php' (length=37)  'PHP\_SELF' => **string** '/public/index.php' (length=17)  'HTTP\_HOST' => **string** 'localhost:8000' (length=14)  'HTTP\_CONNECTION' => **string** 'keep-alive' (length=10)  'HTTP\_CACHE\_CONTROL' => **string** 'max-age=0' (length=9)  'HTTP\_UPGRADE\_INSECURE\_REQUESTS' => **string** '1' (length=1)  'HTTP\_USER\_AGENT' => **string** 'Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36' (length=105)  'HTTP\_ACCEPT' => **string** 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8' (length=85)  'HTTP\_ACCEPT\_ENCODING' => **string** 'gzip, deflate, br' (length=17)  'HTTP\_ACCEPT\_LANGUAGE' => **string** 'fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7' (length=35)  'HTTP\_COOKIE' => **string** 'PHPSESSID=5es0t2nb7flsrjd6mmjpn3a6am' (length=36)  'REQUEST\_TIME\_FLOAT' => float 1551655944.0377  'REQUEST\_TIME' => int 1551655944 |

Voici à quoi devrait ressembler votre dépôt à la fin de cette étape : <https://github.com/le-campus-numerique/myframework/tree/0.0.3>

## 

## Étape 4 : Routeur

On corse un peu les choses et on passe maintenant au routeur. Mais qu'est-ce qu'un routeur ? C’est une classe qui nous permettra de rediriger une requête vers la bonne action d'un contrôleur.  
  
Dans un premier temps, nous allons voir la classe Route. Une route se définit par un nom, un chemin, des paramètres, le contrôleur et l'action vers lequel elle pointe.

Voici la classe [Route](https://github.com/le-campus-numerique/myframework/blob/0.0.4-1/core/Router/Route.php).

Passons maintenant à la classe Router, cette dernière stockera l'ensemble des routes de notre application web.

Finaliser l’implémentation de la classe [Router](https://github.com/le-campus-numerique/myframework/blob/0.0.4-1/core/Router/Router.php)

Pour mieux contrôler le bon fonctionnement de l’implémentation de la classe Router, j'ai créé un [contrôleur](https://github.com/le-campus-numerique/myframework/blob/0.0.4-1/src/Controller/TestsController.php) qui hérite de [AppController](https://github.com/le-campus-numerique/myframework/blob/0.0.4-1/src/Controller/AppController.php) lui-même héritant de [Controller](https://github.com/le-campus-numerique/myframework/blob/0.0.4-1/core/Controller.php). Et j’ai aussi modifié notre [point d’entrée](https://github.com/le-campus-numerique/myframework/blob/0.0.4-1/public/index.php) pour ajouter de nouvelles routes !

### Procédure

* On instancie notre routeur (public/index.php)
* On ajoute de nouvelles routes dans notre routeur ($router->addRoute(...))
* On appelle la méthode Router::getCurrentRoute pour récupérer notre route associée à la requête de l'utilisateur
  + On va tester chaque route pour retourner la première qui correspond à la requête
  + La méthode Route::match permet dans un premier temps de tester si c'est la bonne route et de récupérer les paramètres de la requête
* La méthode Route::call appelle dynamiquement l'action du contrôleur

Si tout se passe bien, en visitant la page <http://localhost:8000/tests/foo> le message Hello world ! devrait apparaître.

Tout comme en visitant la page <http://localhost:8000/tests/bar/good> le message good devrait apparaître.

Pour en savoir plus sur le Router voici un tutorial : [tutoriel de Grafikart](https://www.grafikart.fr/tutoriels/router-628).

Voici à quoi devrait ressembler votre dépôt à la fin de cette étape : <https://github.com/le-campus-numerique/myframework/tree/0.0.4>

## Étape 5 : Contrôleurs

Le contrôleur frontal (celui dans le dossier core) est une classe abstraite qui nous permettra de simplifier le développement de notre application. Il évoluera au fur et à mesure du projet. Par conséquent, chaque contrôleur dans src/Controller sera une classe héritée du contrôleur frontal en héritant elle-même de AppController.

Compléter le [TestsController](https://github.com/le-campus-numerique/myframework/tree/0.0.5-1), l’idée étant que lorsque l’on visitera l’url <http://localhost:8000/tests/redirection/redirected> on arrive sur l’url <http://localhost:8000/tests/bar/redirected> avec le message redirected qui s’affiche !

Regarder les fichiers [index.php](https://github.com/le-campus-numerique/myframework/commit/443e22534ab4bc91ecda91282d98dafc7637497f" \l "diff-8fda5307a067372337ec4b9adb16b55f) / [class Route](https://github.com/le-campus-numerique/myframework/commit/443e22534ab4bc91ecda91282d98dafc7637497f" \l "diff-52bb8cfed8283e0ba153dfeea7e6b172) / [class Router](https://github.com/le-campus-numerique/myframework/commit/443e22534ab4bc91ecda91282d98dafc7637497f" \l "diff-e9b4d20433b8663c5f74f147f25d3785) / [class Controller](https://github.com/le-campus-numerique/myframework/commit/443e22534ab4bc91ecda91282d98dafc7637497f" \l "diff-e1eac6247d2ee015e2b1b26793caf13e) ils ont été modifiés.

D’ailleurs une erreur traîne peut-être dans ces modifications …?

Voici à quoi devrait ressembler votre dépôt à la fin de cette étape : <https://github.com/le-campus-numerique/myframework/tree/0.0.5>

## 

## Étape 6 : Vues

Nous allons maintenant intégrer blade à notre projet, pour simplifier l'écriture des vues.  
  
Dans un premier temps, nous allons installer la [dépendance](https://github.com/duncan3dc/blade) avec composer.  
  
Une fois que votre dossier vendor est à jour, il faut paramétrer blade, et nous allons faire cela dans notre contrôleur frontal. Nous allons donc implémenter une nouvelle méthode de notre contrôleur frontal qui nous permettra d'afficher une vue simplement. La méthode render !

Celle-ci générera la vue via la dépendance blade et l’affichera. Il ne nous restera plus qu'à créer les templates de vues et passer les bons paramètres dans notre Contrôleur d’application ! (TestsController)

Cette fois pas d’aide, à vous de travailler pour faire afficher les messages dans les vues et non plus dans le contrôleur !

Voici à quoi devrait ressembler votre dépôt à la fin de cette étape : <https://github.com/le-campus-numerique/myframework/tree/0.0.6>

## Étape 7 : Authentification et persistance de l’identification

Nous allons créer un nouveau contrôleur : UsersController celui-ci contiendra 3 fonctions liées à 2 vues :

* **Fonctions :**
  + **index()** : Vérifie si une session valide existe, sinon redirige sur login() avec un message d’erreur via l’AuthComponent.
  + **login()** : Devra vérifier le couple de donnée email / password puis générer une nouvelle session d’authentification via l’AuthComponent, sinon générera un message d’erreur.
  + **logout()** : Détruit les données d’authentification et redirige sur login() avec un message de succès via l’AuthComponent.
* **Vues :**
  + **index.blade.php** : Affichera des données sensibles, comme par exemple le nom de votre animal de compagnie.
  + **login.blade.php** : Affichera un simple formulaire en POST avec comme champs :
    - Email
    - Password
    - Submit
  + **logout.blade.php** : Avons nous vraiment besoin de gérer une vue pour le logout ?

Les fonctionnalitées de création, vérification et destruction de session d’authentification devront se faire dans une classe à part : **AuthComponent** celle-ci contiendra **3 fonctions statiques** ce qui nous permettra de sécuriser selon nos besoins nos contrôleurs :

* **::checkAuthenticated()** : Vérifie si une session d'authentification existe
* **::create()** : Génère une nouvelle session d’authentification
* **::delete(Route $route)** : Détruit la session d’authentification courante et redirige vers la $route

Nous utiliserons comme couple d’email / password les informations suivantes afin de pouvoir se connecter :

* Email : [le-campus-numerique@in-the-alps.fr](mailto:le-campus-numerique@in-the-alps.fr)
* Password : LeCampusNumerique@2019

### Procédure 1 :

* Un utilisateur essaie d’accéder à la page <http://localhost:8000/users/index> sans session d’authentification valide.
* La fonction index du contrôleur UsersController test la session d’authentification
* Celle-ci n’est pas valide et donc notre utilisateur est redirigé vers le login() avec un message d’erreur.

### Procédure 2 :

* Un utilisateur valide correctement le formulaire de la page <http://localhost:8000/users/login>
* La fonction login() vérifie les données $\_POST de la Request, les valides et génère une nouvelle session d’authentification : $\_SESSION
* L’utilisateur est redirigé vers la fonction index()

### Procédure 3 :

* Un utilisateur avec une session d’authentification demande à se déconnecter
* La fonction logout() détruit toutes les données de session d’authentification et redirige l’utilisateur sur login() avec un message de succès

À partir de maintenant fini les exemples de code terminé ! Voici tout de même de l’aide pour s’inspirer d’un existant !

### Ressources :

<https://www.grafikart.fr/tutoriels/session-start-825>

<http://php.net/manual/fr/reserved.variables.session.php>

Nous allons maintenant mettre en place la persistance de l’identification ! Assurez-vous d’être connecté et d’avoir accès à la page <http://localhost:8000/users/index>. Fermez votre navigateur et ouvrez le de nouveau sur la page <http://localhost:8000/users/index> ! Vous n’êtes plus connecté…

Avec les ressources ci-dessous régler le problème de la persistance de l’identification. Nous voulons au maximum, devoir nous reconnecter une fois par semaine suite à une perte de session d’identification.

### Ressources :

<https://developers.google.com/web/tools/chrome-devtools/manage-data/cookies>

<https://developer.mozilla.org/fr/docs/Outils/Inspecteur_de_stockage>

<https://www.grafikart.fr/tutoriels/session-start-825>

<http://php.net/manual/fr/function.setcookie.php>

## 

## Étape 8 : classe Response

À ce stade, votre petit framework devrait être fonctionnel. Mais nous allons continuer de l'optimiser. Rappelez-vous, nous avons implémenté la classe Request car dans la théorie, le client envoie une requête qui est interprétée par le serveur web, et ce dernier renvoie une réponse, le plus souvent html.

### Procédure :

* Le framework reçoit une requête
* Le routeur pointe vers la bonne action d'un contrôleur
* Le contrôleur renvoie une réponse
* Notre index transmet la réponse au client

Nous allons créer autant de classes héritant de Response qu'il y a de réponse différente.

* Créer la classe **Reponse**
* Créer la classe **HtmlResponse** qui hérite de la classe **Response**
* Créer la classe **RedirectResponse** qui hérite de la classe **Response**
* Créer la classe **JsonResponse** qui hérite de la classe **Response**

### Ressources :

<https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP>

<https://github.com/sbstjn/php-kickstart/blob/master/libs/Response.class.php>

<https://github.com/terdia/json-response>

## 

## Étape 9 : Modèles

Nous entrons maintenant dans l'une des parties les plus complexes de notre framework maison : les interactions avec une base de données.

Ce que nous allons créer est un petit ORM ([Object Relational Mapping](https://en.wikipedia.org/wiki/Object-relational_mapping)). Le but est de centraliser tous nos traitements de manière plus générique. En gros, je n'ai pas envie de devoir écrire plusieurs fois une requête du style UPDATE ma\_table SET mon\_chhamp = ma\_valeur, pourquoi ne pas implémenter une méthode pour n'écrire qu'une seule fois cette requête et qu'elle s'adapter à toutes les situations ?

Modifier votre code afin de pouvoir gérer un [CRUD](https://fr.wikipedia.org/wiki/CRUD) sur UsersController, voici les informations du schéma de la base de donnée pour la table users (Id: int auto\_increment, Nom: varchar(32), Prénom: varchar(32), Email: varchar(256), Password: varchar (256)).

### Ressources :

<https://github.com/noetix/Simple-ORM>

<https://www.grafikart.fr/tutoriels/tp-database-566>

<https://www.grafikart.fr/tutoriels/tp-tables-567>

## Pour aller plus loin

### Configurations :

Nous allons utiliser un fichier de type yaml pour lister les routes de notre projet. Et pour cela, nous allons utiliser une librairie, ou plus exactement le [composant yaml du framework Symfony](https://symfony.com/doc/current/components/yaml.html).

### Cache simple :

Nous pouvons améliorer notre système de Routeur, actuellement nous calculons à chaque appel si les routes existent à travers des [regex](http://php.net/manual/fr/ref.pcre.php) ce qui n’est pas le plus rapide.

Créer un dossier **tmp/cache** à la racine de votre projet et faites générer un fichier contenant la liste des routes avec leurs informations.

* [serialize](http://php.net/manual/fr/function.serialize.php) à la génération
* [unserialize](http://php.net/manual/fr/function.unserialize.php) à la lecture

Régénération à chaque changement du fichier **config/routes.php**