S5 – PHPA503 – OSS PHP avancé et architecture des frameworks

TP: Framework PHP – Partie 2

1 TP

1.1 Rendu final

Cette partie du TP sera notée. Votre rendu devra contenir les sources de votre framework ainsi qu'un fichier texte à la racine avec des réponses aux différentes questions posées tout au long du TP (un résumé des questions est disponible sur la dernière page de ce document).
Pour résumer votre fichier final devra contenir :

- Les sources de votre framework
- Un dossier *install* à la racine avec toutes les données MySQL
- Un fichier texte TP.txt à la racine avec les réponses aux 5 questions de ce TP

1.2 Notation

- La note principale sera donnée sur votre code, attention à bien tout commenter correctement et respecter les conventions.
- Une note avec un coefficient moindre sera donnée pour les réponses aux 5 questions.
- *Optionnel : un oral peut être organisé avec ce TP selon l'avancement.*

2 Construction du MVC

2.1 Introduction

La base de notre applicatif est en place. Nous allons nous appuyer sur l'architecture déployée lors de la partie 1, cette partie doit être commune à chacun.

Mettre en place une architecture MVC est une tâche compliquée dans le sens où l'on peut toujours améliorer son architecture afin de répondre à de nouvelles problématiques.

Le but de ce TP va être de commencer par une architecture MVC basique, puis de venir au fur et à mesure identifier les limites de notre framework pour ensuite enrichir son code.

2.2 Les contrôleurs (2 pts)

Nous allons commencer par construire notre premier contrôleur.

Actuellement, nous référençons directement notre vue dans notre fichier routes.php

Le but du MVC est que ce soit le contrôleur qui fasse le lien entre la vue et notre modèle. L'utilisateur va donc demander une certaine page via le protocole http, le serveur va devoir interpréter cette demande et retourner le bon résultat à l'utilisateur.

Pour ce faire, nous avons mis en place un système de routing, qui permet d'analyser l'URI demandée (dans notre Application.php) puis de diriger en conséquence l'utilisateur vers la bonne vue. Notre but sera de rajouter une étape afin que notre application appelle notre contrôleur qui lui-même appellera la vue.

- ✓ Nous allons commencer par modifier le fichier routes.php pour au lieu de renseigner la vue, inclure le chemin vers notre classe contrôleur.
- ✓ Il va aussi falloir modifier le fichier Application.php afin d'appeler notre contrôleur selon l'URI demandée.

Ne reste alors plus qu'à construire notre contrôleur, il est important de toujours avoir une classe de base dont on viendra hériter.

- ✓ Créer la classe de base Controller.php dans notre dossier vendor/root
- ✓ Nous allons faire au plus simple pour commencer. Son but va être de charger un template dans le constructeur, puis d'avoir une fonction de rendue pour afficher ce template.

Nous voulons ici rajouter une étape pour l'affichage de notre vue, mais le principe est le même. Soit au lieu d'inclure directement notre vue dans notre Application.php, nous voulons le faire dans notre contrôleur.

- ✓ Créer la classe de notre contrôleur dans le module Oss qui viendra hériter de Controller.php

A vous de jouer ensuite pour correctement appeler nos 2 nouvelles classes dans notre framework et afficher notre vue.

2.3 La vue (1 pts)

Charger à chaque fois une vue avec toutes les balises HTML n'est pas l'idéal. Le jour où l'on veut modifier le header ou footer, il va falloir changer toutes nos vues ?

Le principe est le même que d'habitude, nous allons devoir découper nos vues.

- ✓ Créer un nouveau dossier template dans le dossier Oss/view
- ✓ Créer deux vues dans ce dossier : header.php et footer.php
- ✓ Modifier notre vue par défaut pour n'afficher que le contenu central
- ✓ Revoir notre fonction de rendue dans le contrôleur afin de correctement appeler nos 3 fichiers vues.

Cette première étape nous permet déjà de n'avoir qu'un seul fichier à modifier si jamais nous voulons par exemple ajouter un fichier css ou js à notre applicatif.
Tout comme le contrôleur, cela est une gestion fondamentale de nos vues. Nous pourrons revenir par la suite sur cette partie afin d'identifier, comment nous pouvons l'améliorer.

2.4 Le modèle (3 pts)

2.4.1 Connexion à la base de donnée (2 pts)

Notre première étape sera la connexion à notre base MySql.

- ✓ Pour cela il faut commencer par créer une nouvelle base, nous n'allons pas créer cette base directement dans Mysql mais créer un fichier .sql dans un dossier à la racine de notre projet : *install*. Ce fichier nous permettra de peupler notre base de données par la suite.

- ✓ Pour que notre framework se connecte à notre base de données, nous allons ajouter un nouveau fichier dans le dossier configuration dans lequel nous allons déclarer toutes les informations nécessaires à la liaison avec notre base de données.

- ✓ Il suffit ensuite de créer une classe qui va se connecter à notre base de données Mysql. Nous pourrons la placer dans vendor/Root, soit directement, soit dans un sous-dossier.

- ✓ *Optionnel (+0.5)* : Créer une classe abstraite dont va hériter notre classe de connexion à MySql, cela permettra d'être flexible si jamais vous voulez changer de SGBD.

Nous avons notre connexion à MySql, la classe doit contenir un constructeur, une fonction pour effectuer l'ouverture de la connexion et éventuellement une autre pour se déconnecter et enfin un getter qui retournera notre connexion à la base de données.

Il faudra aussi bien penser à venir charger notre classe dans notre framework.

2.4.2 Model.php (1 pts)

Tout comme notre contrôleur, nous aurons besoin d'une classe de base pour nos modèles. Cette classe ne fera pas grand-chose pour le moment à part initialiser une variable de classe qui contiendra la connexion à notre base de données.

- ✓ Créer la classe Model.php dans vendor\Root

Vous pouvez aussi ajouter le code ci-dessous dans votre Application.php, il permettra de venir charger tous les modèles de vos modules de manière automatique au chargement de votre framework.

```php
/**
 * Load all models
 *
 * @return Application
 */
private function loadModels()
{
    // Load all the model files we have to /modules/***/model
    if( $dir = opendir( MODPATH ) ){
        while( $module = readdir( $dir ) ) {
            if (is_dir(MODPATH . DS . $module) && $module !== '.' && $module !== '..')
{
                $modelPath = MODPATH . DS . $module . DS . 'model';
                if (is_dir($modelPath) && $modelDir = opendir($modelPath)) {
                    while( false !== ($file = readdir( $modelDir )) ){
                        if( !is_dir($modelPath . DS . $file) && $file !== '.' && $file !== '..' ){
                            require_once $modelPath . DS . $file;

                        }
                    }
                }
            }
        }
        closedir($dir);
    } else {
        throw new \Exception("Impossible to access the model path.", 1);
    }
}
```

Vous n'aurez alors plus qu'à appeler cette fonction juste avant d'exécuter la fonction *call*.

2.5 Notre premier module (3 pts)

De base, notre framework n'a pas besoin de modèle. Nous allons donc créer notre premier module d'exemple afin de créer notre premier modèle puis construire autour tout le nécessaire pour afficher les données sur un navigateur.

Cela nous permettra de créer notre premier modèle mais surtout de voir par la suite qu'elles sont les parties de notre framework qu'il va falloir améliorer. C'est en utilisant une application que l'on en découvre ses limites.

2.5.1 Le module User (1 pts)

Pour l'exemple, nous allons créer un module User.

- ✓ Créer un module User avec 3 dossiers, soit notre MVC.
- ✓ Dans notre fichier SQL ajouter à notre base de données une table user avec quelques champs tels que le nom et le prénom.
- ✓ Insérer dans notre base de données des valeurs d'exemples, soit une dizaine de clients de test.

Il ne nous reste alors plus qu'à créer nos 3 fichiers, soit notre modèle et notre vue qui seront gérés par un contrôleur.

2.5.2 Le modèle : user (1 pts)

Notre modèle va héritier de notre classe de base.
Nous allons vouloir dans un premier temps afficher la liste de tous les utilisateurs.

- ✓ Créer une classe userModel qui va hériter de notre classe de base. Ne pas oublier d'ajouter une fonction qui retourne tous les utilisateurs.

2.5.3 Terminer notre MVC (1 pts)

- ✓ Maintenant que notre modèle est créé, il ne reste plus qu'à créer une vue, qui affichera tous les noms et prénoms des utilisateurs enregistrés.

Pour l'instant nous n'avons aucun lien entre notre modèle et notre vue.

- ✓ A vous de renseigner un nouveau chemin dans votre fichier de configuration des routes, puis de créer un contrôleur qui va créer ce lien, à l'image de notre premier contrôleur avec juste une étape en plus : initialiser notre modèle.

3 Amélioration de notre MVC

Nous avons construit jusque-là une base de framework avec un premier module d'exemple. Vous aurez peut-être déjà constaté des « limites » à notre architecture.

Cette partie va nous permettre de revenir sur certains aspects et de mettre en pratique les design pattern que nous avons vus en cours.

3.1 Ajout d'un nouveau modèle (1,5 pts)

Pour la suite, nous allons avoir besoin de voir comment s'agencent les modèles entre eux, pour cela nous allons créer un nouveau modèle : Adresse

- ✓ Créer un nouveau modèle dans le module User : addressModel
- ✓ A la suite du nom et du prénom dans notre listing utilisateurs, nous voudrons aussi afficher l'adresse du client.

3.2 Interconnexion avec la base de données : questions

Question 1 : Si vous n'avez mis en place aucun système pour gérer les instances de votre classe qui vous permettra de vous connecter à la base de données, nous constatons un problème, lequel ?

Question 2 : Donnez une solution pour éviter ce problème, puis rajouter cette fonctionnalité à votre framework. Plusieurs solutions sont possibles pour solutionner ce problème, mais comportent aussi des limites, décrivez la principale limite à la solution proposée.

3.3 Interconnexion avec la base de données (1 pts)

Suite aux réponses aux questions 1 et 2 :

- ✓ Rajoutez la fonctionnalité choisie à votre framework.

3.4 Lier nos modèles User et Address (1,5 pts)

Notre classe User et notre classe Address se doivent d'être « liées » ensemble, une adresse est un objet qui appartient en effet à un utilisateur.

Toutefois, il faut faire attention de ne pas « trop les lier » car si on modifie la classe address, il ne faudrait surtout pas toucher notre classe user. (imaginez avec 20 classes ce que cela peut donner …)

Nous allons pour cela utiliser l'injection de dépendance.

Le principe est simple, si une classe a besoin d'un autre objet, que ce soit dans son constructeur où n'importe qu'elle méthode alors elle prend l'instance de l'objet directement en paramètre et ne s'occupe surtout pas de l'instancier elle-même.

- ✓ Créer une nouvelle page dans notre framework afin de créer un nouvel utilisateur.
- ✓ Il va falloir pour cela créer un formulaire avec les champs obligatoires pour l'utilisateur ainsi que pour son adresse.
- ✓ A vous de compléter votre module user pour arriver à ce résultat.

Question 3 : L'injection de dépendance ne suffit souvent pas à elle seule à ce que nos classes soient vraiment indépendantes. Expliquez comment vous feriez pour améliorer votre architecture.

- ✓ *Optionnel (+2)* : Mettez en place les améliorations décrites dans la question 3.

3.5 A vous de jouer ! (3 pts)

Question 4 : Décrivez une amélioration possible à apporter à la gestion des vues

Question 5 : Décrivez jusqu'à 3 fonctionnalités supplémentaires à apporter à votre framework.

- ✓ Mettez en place les améliorations décrites dans les parties 4 et 5

4 Résumé des questions du TP (4 pts)

Question 1 : Si vous n'avez mis en place aucun système pour gérer les instances de votre classe qui vous permettra de vous connecter à la base de données nous constatons un problème, lequel ?

Question 2 : Donnez une solution pour éviter ce problème, puis rajouter cette fonctionnalité à votre framework. Plusieurs solutions sont possibles pour solutionner ce problème, mais comportent aussi des limites, décrivez la principale limite à la solution proposée.

Question 3 : L'injection de dépendance ne suffit souvent pas à elle seule à ce que nos classes soient vraiment indépendantes. Expliquez comment vous ferriez pour améliorer votre architecture.

Question 4 : Décrivez une amélioration possible à apporter à la gestion des vues

Question 5 : Décrivez jusqu'à 3 fonctionnalités supplémentaires à apporter à votre framework.