Programmation Web sur le Serveur

Mini projet Tweeter (partie 2 - Le contrôle)

Amine boumaza

amine.boumaza@univ-lorraine.fr

Le contrôleur

L'objectif d'un contrôleur pour une application Web est de réaliser une fonctionnalité donnée. Un contrôleur est dans ce cas, l'instance d'une classe dont certaines méthodes réalisent complètement une fonctionnalité.

Ces méthodes doivent avoir accès a toutes les données nécessaires (modèles et paramètres GET ou POST) afin de réaliser la fonctionnalité. De manière générale l'algorithme d'une méthode d'un contrôleur est souvent comme suit :

- Récupérer les données nécessaires pour la fonctionnalité
- Exécuter le traitement nécessaire
- Créer la vue liée à la fonctionnalité et l'afficher.

A ce stade, les contrôleurs ne créent pas de vues, on se contentera d'un affichage simple (example, **`var_dump`** ...).

Implantation du contrôleur

L'objectif de cette étape est d'implanter seulement la première fonctionnalité, "afficher la liste des Tweets".

Le contrôleur qui devra réaliser cette fonctionnalité est la classe **`tweeterapp\control\TweeterController`**. Cette classe hérite de la classe abstraite **`mf\control\AbstractController`**.

Attention : le fonctionnement des méthodes suivantes repose sur le bon fonctionnement des modèles de la partie précédente.

- Télécharger les fichiers TweeterController.php et AbstractController.php et bien les étudier. Les enregistrer dans les bons répertoires (selon l'espace de noms ou ils appartiennent)
  - Coder la méthode **`viewHome`** en suivant l'algorithme donné dans le code.
  - Tester ce contrôleur, en ajoutant le fragment de code suivant dans le script principal :

```php
<?php

/* configuration d'Eloquent (cf partie 1) */

$ctrl = new tweeterapp\control\TweeterController();
echo $ctrl->viewHome();
```

Le routage des URLs

Le rôle principal d'un routeur d'URL est d'aiguiller le URLs vers les bons contrôleurs afin d'exécuter les fonctionnalités demandées. Il permet aussi de faciliter certaines fonctionnalités annexes d'un *framework*. Par exemple il fournit un moyen d'exécuter une route sans qu'elle soit explicitement demandée par un utilisateur, ou de récupérer l'URL d'une fonctionnalité afin de construire les liens hypertexte par exemple.

Le principe du routage d'URL est de diriger l'exécution de l'application vers une fonctionnalité donnée en fonction de l'URL demandée. À chaque fonctionnalité correspondra donc une URL unique. L'URL est analysée par le **routeur d'URL** et ce dernier déclenche l'exécution de la méthode du contrôleur pour la fonctionnalité.

*Réaliser la classe **`Router`***

Les trois fonctionnalités abordée à ce stade ont les URL suivantes :

- Afficher la liste des Tweets : **`http:// ... /Tweeter/main.php/home/`** ;
- Afficher un Tweet donnée (ici d'identifiant 13) : **`http:// ... /Tweeter/main.php/view/?id=13`** ;
- Afficher les Tweets d'un utilisateur donné (ici d'identifiant 2) : **`http://.../Tweeter/main.php/user/?id=2`**

La classe **`mf\router\Router`** hérite de la classe abstraite **`mf\router\AbstractRouter`** qui définie 4 méthodes **`addRoute`**, **`setDefaultRoute`**, **`run`** et **`urlFor`** en plus des attributs statiques **`$routes`** et **`$aliases`**.

- Télécharger la classe abstraite AbstractRouter.php et l'étudier. Les commentaires donnés fournissent les algorithmes des méthodes à coder.

L'attribut statique **`$routes`** est un tableau qui va stocker les routes. Chaque route est elle même un tableau de 2 éléments de la forme : **`[ contrôleur, méthode ]`**. Ou **`contrôleur`** est le nom de la classe qui fournit **`méthode`** qui réalise une fonctionnalité. Chaque route est stockée dans le tableau sous la clé donnée par son URL.

Note : URL est utilisé ici par abus de langage, en réalité on ne considère que la partie de l'URL qui suit **`main.php`**, le **`path_info`**.

Le second attribut **`$aliases`**, lui aussi statique, contient les alias des routes. Ces alias serviront à nommer les routes a fin de facilité leur utilisation. Chaque élément du tableau **`$alias`** est une URL stockée sous une clé qui est l'alias de la route.

La **route par défaut** est la fonctionnalité exécutée si l'utilisateur ne fournie pas d'URL (s'il exécute **`main.php`** tout simplement). Elle est a définir par le concepteur. Dans notre cas c'est la fonctionnalité "afficher l'ensemble des Tweets".

1. Coder la classe **`mf\router\Router`** qui concrétise la classe **`mf\router\AbstractRouter`** ;
2. Coder un constructeur qui appelle le constructeur de **`AbstractRouter`** ;
3. Coder la méthode **`addRoute`** ;
4. Coder la méthode **`setDefaultRoute`** ;
5. Vérifier le bon fonctionnement de la méthode comme suit :

Le but de la méthode **`addRoute`** est d'enregistrer les routes (URL) possibles et leurs alias lors du démarrage de l'application. Elle sera invoquée depuis le script principal comme suite :

```php
<?php

/* configuration d'Eloquent (cf partie 1 du projet ) */
```

```php
$router = new \mf\router\Router();

$router->addRoute('maison',
                  '/home/',
                  '\tweeterapp\control\TweeterController',
                  'viewHome');

router->setDefaultRoute('/home/');

/* Après exécution de cette instruction, l'attribut statique $routes et
   $aliases de la classe Router auront les valeurs suivantes: */

print_r(Router::$routes);

/* Affichera */
Array
(
   [/home/] => Array
      (
         [0] => \tweeterapp\control\TweeterController
         [1] => viewHome
      )
)

print_r(Router::$aliases);

/* affichera */
Array
(
       [maison] => /home/
       [default] => /home/
)
```

Exécution des routes

L'objectif de la méthode **run** est de déclencher la fonctionnalité demandée. Elle utilise les informations stockées dans l'attribut **$http_req** (voir code de **AbstractRouter**) qui lui permet de savoir quelle URL a été demandée. En fonction de l'URL elle créer une instance du contrôleur et invoque la méthode stockés dans le tableau des routes. S'il n'y à pas de **path_info** dans l'URL ou si elle ne correspond pas à une route connue, la route par défaut est utilisée.

1. Coder la méthode **run** ;
2. Dans le script principal **main.php**, ajouter quelques routes et tester leur bon enregistrement dans le tableau ;
3. Tester le fonctionnement de la méthode **run** avec le code suivant :

```php
<?php

...

$router = new \mf\router\Router();

$router->addRoute('home', '/home/', '\tweeterapp\control\TweeterController', 'viewHome');

$router->setDefaultRoute('/home/');

$router->run();
```

Maintenant que la fonctionnalité qui affiche la liste des Tweets fonctionne, nous allons ajouter les deux autres fonctionnalités restantes : afficher 1 Tweet et afficher les Tweets d'un utilisateur.

- Pour chacune des deux, suivre le schéma suivant :
    1. Dans le contrôleur **\tweeterapp\control\TweeterController**

*coder la méthode qui réalise la fonctionnalité (à l'image de **`viewHome`**).*

1. Dans le script principal, ajouter la route de cette fonctionnalité.

Exécution des routes nommées

Dans certaines situation, il est nécessaire d'exécuter une route sans quelle soit explicitement demandée par l'utilisateur (sans rentrer d'URL). Par exemple, si un utilisateur ne fourni pas un mot de passe correcte il faut le diriger vers une fonctionnalité spécifique (pour gérer cette situation). Dans ce cas l'utilisateur ne fourni pas d'URL mais c'est l'application qui en exécute une. C'est dans ce cas de figure que l'alias rentre en jeu. L'application exécute une route en fournissant son alias.

Afin d'exécuter une route en utilisant son alias il faut ajouter une méthode à la classe **`Router`**, qui prend en paramètre l'alias.

Cette méthode sera une méthode statique car elle accède au tableau statique **`$routes`** et sera exécutée sans nécessiter une instance de la classe **`Router`**.

- Dans la classe **`Router`** ajouter la méthode statique **`executeRoute`** qui prend en paramètre une chaîne de caractères (l'alias d'une route) et qui exécute cette route (a l'image de la méthode **`run`**)

Récupérer l'URL d'une route nommée

Il est souvent nécessaire d'avoir l'**`URL`** d'une fonctionnalité donnée à différentes parties de l'application. Par exemple si on souhaite réaliser un lien hypertexte qui exécute une fonctionnalité de l'application, il faut au moment de générer ce lien (au moment d'écrire la balise **`HTML <a href="..."> lien </a>`**) avoir l'**`URL`** cible.

Pour cela le routeur fourni souvent une méthode qui permet a partir de l'alias d'une route de récupérer sont URL complète.

- Coder la méthode **`urlFor`** de la classe **`Router`**.
- Tester cette méthode en l'utilisant pour créer les liens hypertextes qui permettent d'afficher un Tweet et les Tweets d'un utilisateur (dans le bloc HTML affiché par le contrôleur)

Valider votre code

Pour valider votre code avec les test unitaires **`phpunit`**, vous pouvez télécharger les scripts suivants :

- RouterTest.php : les tests unitaires pour la classe **`Router`** ;

Note Il faut lancer les tests depuis le répertoire **`Tweeter`**. Par example :

```
terminal> phpunit --color --globals-backup tests/RouterTest.php
```

Votre Progrès

Avancement :

Pour avoir une idée se a quoi doit ressembler l'application aller ici.

A ce stade vous devez avoir les fichiers suivants :

```
Tweeter
├── composer.json
├── composer.lock
├── conf
│   ├── config.ini
│   └── tweeter.sql
├── html
├── main.php
├── src
│   ├── mf
│   │   ├── control
│   │   │   └── AbstractController.php       [Nouveau]
│   │   ├── router
│   │   │   ├── AbstractRouter.php           [Nouveau]
│   │   │   └── Router.php                   [Nouveau]
│   │   └── utils
│   │       ├── AbstractClassLoader.php
│   │       ├── AbstractHttpRequest.php
│   │       ├── ClassLoader.php
│   │       └── HttpRequest.php
│   └── tweeterapp
│       ├── control
│       │   └── TweeterController.php        [Nouveau]
│       └── model
│           ├── Follow.php
│           ├── Like.php
│           ├── Tweet.php
│           └── User.php
├── tests
│   ├── ClassLoaderTest.php
│   ├── dummy
│   │   └── classname
│   │       └── Name.php
│   ├── HttpRequestTest.php
│   └── RouterTest.php                       [Nouveau]
└── vendor
```