Programmation Web sur le Serveur

Mini projet Tweeter (partie 3 - les vues)

Amine boumaza

amine.boumaza@univ-lorraine.fr

Les vues

L'objectif de la vue pour une application Web est de réaliser l'affichage du résultat d'une fonctionnalité donnée. La vue est dans ce cas l'instance d'une classe dont certaines méthodes retournent les fragments HTML qui seront combinés pour former une page entière sur le client.

Ces méthodes doivent avoir accès à certaines données nécessaire pour réaliser le résultat de la fonctionnalité. Par exemple si on souhaite afficher la liste des Tweets, la méthodes qui crée le fragment HTML pour afficher cela, devra avoir accès à cette information (la liste des Tweets) sous la forme d'un modèle.

La création d'une vue se fait dans le contrôleur. Chaque méthode du contrôleur qui réalise une fonctionnalité crée la bon vue (voir la partie contrôleur). Des exemples de vues sont disponible dans la liste des fonctionnalités.

Implantation de la vue

Les vues de l'application sont générées par la classe TweeterView.php. Cette classe concrétise AbstractView.php.

Elle défini différents méthodes dont le nom commence par **`render...`** et dont l'objectif est de retourner un fragment de code **`HTML`**. Chacune des méthodes est chargée de rendre une partie de l'interface.

1. Télécharger les fichiers AbstractView.php et TweeterView.php dans les bons espaces de noms.
2. Étudier ces classe sans rien modifier pour l'instant:

   - *Noter les attributs de la classe **`AbstractView`**, comprendre leur rôle.*
   - *Comprendre le rôle de la méthode **`render`** de **`AbstractView`** et des méthodes **`render...`** de la classe **`TweeterView`**.*

Faire le lien avec le contrôleur

Comme évoqué dans la partie contrôleur, l'algorithme d'une méthode du contrôleur est généralement comme suit :

1. Récupérer les données nécessaires pour la fonctionnalité
2. Exécuter le traitement nécessaire
3. Créer la vue liée à la fonctionnalité en lui passant les donnée

4. Afficher la vue.

Le contrôleur crée une instance de la vue (étape 3) en passant les données nécessaires qui doivent être affichées en paramètre au constructeur. Dans le cas de l'application Tweeter, ce paramètre sera selon les cas : un **`Tweet`**, un tableau d'objets **`Tweet`**, un **`User`** ou un tableau de **`User`**.

1. Coder les méthodes **`renderHome`**, **`renderUserTweet`** et **`renderTweet`** de la classe **`TweeterView`**.
    - Par exemple la méthode **`renderTweet`** retourne le bloc HTML qui réalise l'affichage d'un Tweet. Le Tweet en question est disponible dans l'attribut **`$this->data`** (ce dernier est passé au constructeur de **`TweeterView`**)
2. Modifier les méthodes **`viewHome`**, **`viewUserTweet`** et **`viewTweet`** de la classe **`TweeterController`** de sorte qu'elles crées une instance de **`TweeterView`** en lui passant la donnée adéquate.
3. Tester le fonctionnement. Pour cela exécuter les méthodes **`renderHome`**, **`renderUserTweet`** et **`renderTweet`** depuis les contrôleurs associés respectivement : **`viewHome`**, **`viewUserTweet`** et **`viewTweet`** de la classe **`TweeterController`**. Par exemple :

```php
<?php

/* dans le contrôleur viewHome */

public function viewHome(){

        $tweets = \tweeter\model\Tweet::all();
        $vue = new \tweeter\view\TweeterView($tweets);
        echo $v->renderHome();
```

Il faudrait temporairement rendre les méthodes **`render...`** publics pour l'occasion

Élaborer un patron

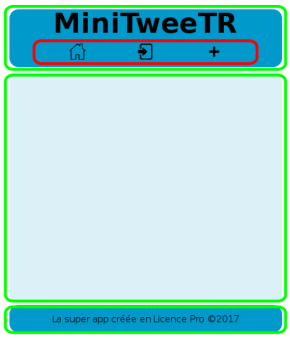
Les vues d'une application Web ont généralement une structure unique, dont le contenu change en fonction de la fonctionnalité réalisés. Cette structure suit un ou plusieurs "patrons" fixés lors de la conception de l'application.

Ces patrons sont définis en utilisant les éléments HTML de haut niveau dans la structure du document qui englobent d'autres élément avec les différents contenus.

La figure ci-contre montre un exemple de patron utilisé dans l'application Tweeter. Les trois boites de couleur verte contiendront du contenu qui va correspondre au différentes vues de l'application. Le contenu de certaines (en haut et en bas) ne changera pas ou très peu en fonction du contexte.

Le document HTML qui correspond au patron de la vue est généralement réalisé par une méthode qui retourne une chaîne de caractère qui contient le document complet. Cette méthode reçoit un sélecteur en fonction duquel la méthode choisira le bon contenu pour chaque bloc (cf. TD afficheur de personne en vue courte et vue détail).

Un exemple de mise en page



Le document HTML correspondant au patron

```
<!DOCTYPE html>
<html lang="fr">
   <head>
      <meta charset="utf-8">
      <title> ... </title>
      <link rel="stylesheet" href=" ... ">
   </head>
   <body>
       <header> ... </header>
       <section>
          ...

       </section>
       <footer> ... </footer>
   </body>
</html>
```

Dans notre cas, le patron est défini dans la méthode **`render`** de la clase **`AbstractView`** et dans la méthode **`renderBody`** de la classe **`TweeterView`**. La première construit le squelette de la page HTML, la seconde celui de l'élément **`<body>`**.

1. coder la méthode **`renderBody`** de la classe **`TweeterView`** ;
   - Elle doit retourner la structure HTML qui réalise une page se rapprochant de la figure ci-dessus ;
   - Les différentes boites reçoivent leur contenu des méthodes **`render...`**. Par exemple, l'entête (**`<header>`**) de la page reçoit son contenu depuis la méthode **`renderHeader`** ;
   - Pour le contenu de la partie centrale, utiliser la valeur du paramètre **`$selector`** pour savoir quelle méthode parmi les trois codées précédemment appeler. Choisir 3 valeur distinctes pour le sélecteur une pour chacune des vue possible ;
2. Modifier les contrôleurs en appelant la méthode **`render`** sur l'instance de la vue crée au lieu des méthodes **`render...`** ;
3. Remettre les 3 méthodes précédentes **`renderHome`**, **`renderUserTweet`** et **`renderTweet`** en privé ;
4. Dans ce qui suit il est possible de s'inspirer de ce fichier de style. Mais ce n'est pas une obligation.

Ajouter les liens

L'objectif de cette étape et d'ajouter sur l'interface généré des liens et des zones sur lesquelles l'utilisateur pourra cliquer pour exécuter des fonctionnalités. Ceci se traduit par l'ajout de liens hypertexte dans le document HTML. Les cibles de ces liens sont des routes de l'application.

Les liens hypertexte sont générés par la méthode **`urlFor`** de la classe **`Router`**. Il sont créés au sein des vues.

1. Réaliser les liens hypertexte qui permettent les interactions suivantes :
   - *Un click sur le texte d'un Tweet exécute la fonctionnalité afficher le Tweet.*
   - *Un click sur l'auteur d'un Tweet, exécute la fonctionnalité afficher les Tweets de cet utilisateur.*
- Réaliser le menu encadré en rouge dans la figure ci-dessus.

Ajouter des fonctionnalités

A ce stade les trois parties Modèle, Vue, Contrôl existent. Pour ajouter une nouvelles fonctionnalité à l'application il suffit d'ajouter une route, un contrôleur associée et une vue.

L'objectif ici est d'ajouter la fonctionnalité rédiger et poster un Tweet (voire fonctionnalités pour le détail) Cette fonctionnalité s'exécute en deux étapes :

1. Afficher un formulaire;

> 1. Écrire une méthode dans la classe **`TweeterView`** qui affiche le formulaire de rédaction, a l'image des méthodes **`render...`** ;
> 2. Ajouter à la méthode **`renderBody`** un cas supplémentaire dans les choix du sélecteur pour afficher ce formulaire ;
> 3. Ajouter une méthode dans la classe **`TweeterController`** qui affiche cette vue. (**Note** : il n'y a pas de donnée a récupérer dans ce contrôleur) ;
> 4. Ajouter la nouvelle route (**`/post/`**) à la table de routage de l'application et l'associer a ce contrôleur ;
> 5. Tester l'affichage du formulaire.

2. transmettre les données du formulaire.

> 1. Ajouter la route (**`/send/`**) à la table de routage et l'associer à une méthode du contrôleur. Cette route sera exécutée par le bouton validation du formulaire.
> 2. La méthode du contrôleur crée un nouveau Tweet et le renseigne avec les données du formulaire (Ne pas oublier de filtrer les données) et l'enregistre dans la base de données. **Note** : pour le moment renseigner l'auteur avec un identifiant existant en BD.

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
│   └── style.css                              [Nouveau]
├── main.php
├── src
│   ├── mf
│   │   ├── control
│   │   │   └── AbstractController.php
│   │   ├── router
│   │   │   ├── AbstractRouter.php
│   │   │   └── Router.php
│   │   ├── utils
```

```
│   │   │   ├── AbstractHttpRequest.php
│   │   │   ├── ClassLoader.php
│   │   │   └── HttpRequest.php
│   │   └── view
│   │       └── AbstractView.php              [Nouveau]
│   └── tweeterapp
│       ├── control
│       │   └── TweeterController.php
│       ├── model
│       │   ├── Follow.php
│       │   ├── Like.php
│       │   ├── Tweet.php
│       │   └── User.php
│       └── view
│           └── TweeterView.php               [Nouveau]
├── tests
│   ├── ClassLoaderTest.php
│   ├── dummy
│   │   └── classname
│   │       └── Name.php
│   ├── HttpRequestTest.php
│   └── RouterTest.php
└── vendor
```