IUT de Montpellier - Programmation Web - TD4
Architecture MVC avancée

Rémi Coletta, Romain Lebreton, Bastien Vialla

Semaines du 13 et 20 Octobre 2014

Aujourd'hui nous allons développer notre site-école de covoiturage. Au fur et à mesure que le projet grandit, nous allons bénéficier du modèle MVC qui va nous faciliter la tâche de conception.

Le but de ce TD est donc d'avoir un site qui propose une gestion minimale des utilisateurs et des trajets proposés en covoiturage. En attendant de pouvoir gérer les sessions d'utilisateur, nous allons développer l'interface "administrateur" du site.

1 Mise en route

Pour ce TD, nous vous passons un squelette de site qui sera à compléter et à enrichir.

Q 1. Récupérez sur `http://www.lirmm.fr/~lebreton/teaching.html` l'archive TD4.zip qui vous servira de base pour ce TD. Décompressez cette archive dans votre `public_html`. **Rentrez vos informations de connexion dans `./config/Conf.php`.**

Si vous utilisez NetBeans, ce qui est conseillé, créez un nouveau projet à partir du répertoire TD4 (File → New Project → 'Php application from existing sources' → sélectionnez le répertoire TD4).

Q 2. En utilisant, l'interface de PhpMyAdmin, créez les deux tables si elles n'existent pas déjà (attention aux majuscules) :

1. une table 'utilisateur' avec 4 champs de type VARCHAR(32) : PRIMARY 'login', 'nom', 'prenom', 'email' dont l'interclassement est `utf8_general_ci` (il s'agit de l'encodage qui sera utilisé par défaut dans vos tables).
2. une table 'trajet' avec 6 champs : PRIMARY INT 'id', VARCHAR(32) 'conducteur', VARCHAR(32) 'depart', VARCHAR(32) 'arrivee', INT 'nbplaces', INT 'prix'. On souhaite que le champ primaire 'id' s'incrémente à chaque nouvelle insertion dans la table. Pour ce faire, sélectionnez pour le champ 'id' la valeur par défaut `NULL` et cochez la case `A_I` (auto-increment). L'interclassement sera toujours `utf8_general_ci`.

2 CRUD pour les utilisateurs

CRUD est un acronyme pour Create Read Update Delete, qui sont les 4 opérations de base de toute donnée. Nous allons compléter notre site pour qu'il implémente toutes ces fonctionnalités.

Le site squelette se navigue à partir de la page principale `TD4/index.php` en passant des paramètres dans l'URL comme par exemple `TD4/index.php?controller=utilisateur&action=readAll`. La page principale `TD4/index.php` contient principalement le code du dispatcher, dont la fonction est de charger le bon contrôleur en fonction des paramètres reçus dans l'URL. Dans notre exemple, nous avons passé en paramètres `controller=utilisateur`, donc le dispatcher chargera le contrôleur `ControllerUtilisateur.php`. Le dispatcher fait lui-même partie du contrôleur dans le modèle MVC car il n'affiche ni page HTML (rôle des vues), ni ne manipule les données (rôle des modèles).

À son tour, le contrôleur chargé va, en fonction de l'action donnée en paramètre dans l'URL, traiter la requête, agir avec le modèle correspondant puis afficher la vue adéquate. Dans notre exemple, l'action est 'readAll' et le contrôleur va donc :

— demander au modèle `ModelUtilisateur.php` de lire tous les utilisateurs de la base de donnée ;
— initialiser une variable `$tab_util` ;
— charger la vue `viewListUtilisateur.php` dont la tâche est d'afficher une belle page HTML avec le contenu de `$tab_util`.

Pour l'instant, notre site ne sait que traiter les actions 'read' (recherche), 'readAll' (liste) et 'insert' (création) pour les utilisateurs.

Q 3. Prendre le temps de vérifier que l'on comprend bien le site squelette donné et son organisation. En cas de doute, relire le TD précédent, Googler la partie du code mystérieuse ou demander à votre professeur.

Q 4. On veut rajouter un comportement par défaut pour la page d'accueil. Modifier le code du dispatcher pour qu'un utilisateur qui arrive sur `TD4/index.php` voit la même page que s'il était arrivé sur `TD4/index.php?controller=utilisateur&action=readAll`.

Si aucun paramètre n'est donné dans l'URL, initialisons les variables `$controller` et `$action` comme si le contrôleur 'utilisateur' et l'action 'readAll' étaient donnés en paramètres. Pour tester si un paramètre 'controller' est donné dans l'URL, utilisez la fonction `isset` sur `$_GET['controller']`. La fonction `isset` teste si une variable a été initialisée.

Désormais, la page `http://infolimon.iutmontp.univ-montp2.fr/~votre_login/TD4/index.php` doit marcher sans paramètre. Notez que vous pouvez aussi y accéder avec l'adresse `http://infolimon.iutmontp.univ-montp2.fr/~votre_login/TD4/` : Apache ouvre directement les pages `index.html` ou `index.php` d'un répertoire si elles existent.

Q 5. Nous souhaitons rajouter l'action 'delete' aux Utilisateurs. Pour cela :

1. Complétez dans le modèle d'utilisateur la fonction `delete($data)`. L'entrée `$data` sera un tableau associatif qui contiendra le login à supprimer dans son champ `$data['login']`. Utilisez pour cela les requêtes préparées de PDO, en suivant l'exemple de `select($data)`.
   Rappel : Ce type d'objet `$data` est celui qui est pris en entrée par la méthode `execute` de `PDO`.
2. Complétez l'action 'delete' du contrôleur d'utilisateur pour qu'il supprime l'utilisateur dont le login est passé en paramètre dans l'URL, initialise les variables `$login` et `$tab_util`, puis qu'il affiche la vue précédemment créée.
3. Complétez la vue `viewDeletedUtilisateur.php` pour qu'elle affiche un message indiquant que l'utilisateur de login `$login` a bien été supprimé. Affichez en dessous de ce message la liste d'utilisateurs contenue dans `$tab_util` comme dans la page d'accueil.
4. Enrichissez la vue de détail `viewFindUtilisateur.php` pour ajouter un lien HTML qui permet de supprimer l'utilisateur dont on affiche les détails.
5. Testez le tout. Quand la fonctionnalité marche, appréciez l'instant.

Q 6. Nous souhaitons rajouter l'action 'update' aux Utilisateurs. Pour cela :

1. Complétez dans le modèle d'utilisateur la fonction `update($data)`. L'entrée `$data` sera un tableau associatif associant aux champs de la table 'utilisateur' les valeurs correspondante à l'utilisateur courant. La fonction doit mettre à jour tous les champs de l'utilisateur dont le login est `$data['login']`.
2. Complétez la vue `viewUpdateUtilisateur.php` pour qu'elle affiche un formulaire identique à celui de `viewCreateUtilisateur.php`, mais qui sera pré-rempli par les données de l'utilisateur courant. Toutes les informations nécessaires seront une fois de plus passées *via* l'URL.
   Indice : L'attribut `value` de la balise `<input>` permet de pré-remplir un champ du formulaire. Notez aussi que l'attribut `readonly` de `<input>` permet d'afficher le login sans que l'internaute puisse le changer.
3. Complétez la vue `viewUpdatedUtilisateur.php` pour qu'elle affiche un message indiquant que l'utilisateur de login `$login` a bien été mis à jour. Affichez en dessous de ce message la liste d'utilisateurs mise à jour contenue dans `$tab_util` comme dans la page d'accueil.
4. Complétez l'action 'update' du contrôleur d'utilisateur pour qu'il affiche le formulaire pré-rempli.
5. Complétez l'action 'updated' du contrôleur d'utilisateur pour qu'il mette à jour l'utilisateur dont le login est passé en paramètre dans l'URL, puis qu'il affiche la vue `viewUpdatedUtilisateur.php` après l'avoir correctement initialisé.
6. Enrichissez la vue de détail `viewFindUtilisateur.php` pour ajouter un lien HTML qui permet de mettre à jour l'utilisateur dont on affiche les détails.
7. Testez le tout. Quand la fonctionnalité marche, appréciez de nouveau l'instant.

3 Vues modulaires

En l'état, certains bouts de code de nos vues se retrouvent dupliqués à de multiples endroits. Par exemple, l'affichage de la liste qui se trouve dans `viewListUtilisateur.php` se retrouve en partie dans `viewCreatedUtilisateur.php`, `viewDeletedUtilisateur.php`, `viewUpdatedUtilisateur.php`, ...

Réorganisons le code pour éviter les redondances :

Q 7. Créer un fichier `TD5/view/header.php` avec au moins le code suivant. Cette en-tête de page sera commune à tout votre site. Vous pourrez la personnaliser plus tard avec, par exemple, une barre de menus renvoyant vers les principales pages du site.

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta charset="UTF-8">
5         <title><?php echo $pagetitle; ?></title>
6     </head>
7     <body>
```

Q 8. Créer un fichier `TD5/view/footer.php` avec au moins le code suivant. Ce pied de page sera commun à tout votre site. Vous pourrez la personnaliser plus tard avec un pied de page comportant votre nom, la date de dernière modification de la page, un lien vers un formulaire de contact ou encore les logos certifiant que votre page HTML est conforme aux standards.

```
1     </body>
2 </html>
```

Q 9. Raccourcir toutes les vues en utilisant `require (ROOT . DS . 'view' . DS . 'header.php')` en début de fichier et `require (ROOT . DS . 'view' . DS . 'footer.php')` en fin de fichier. Initialiser la variable `$pagetitle` en début de vue.

Q 10. Mettre le corps de `viewListUtilisateur.php` dans un fichier séparé `partViewListUtilisateur.php`. Réutiliser ce fichier dans `viewCreatedUtilisateur.php`, `viewDeletedUtilisateur.php`, `viewUpdatedUtilisateur.php`, ...

Q 11. Fusionner `viewCreateUtilisateur.php` et `viewUpdateUtilisateur.php` en une unique page. Mettre à jour le contrôleur en conséquence.
Indice : `<input ... placeholder='Exemple' value='$val'>` affichera 'Exemple' en grisé si `$val` est la chaîne de caractère vide, et pré-remplira avec la valeur de `$val` autrement.

4 CRUD pour les trajets

L'implémentation du CRUD pour les trajets est un code très similaire à celui pour les utilisateurs. Nous pourrions donc copier/coller le code des utilisateurs et changer les quelques endroits nécessaires.

Pour éviter de perdre un temps conséquent à développer le CRUD pour chaque nouvel objet, nous allons le créer automatiquement autant que faire se peut.

4.1 Création d'un modèle générique

Q 12. Commençons par la fonction `selectAll()`. Dans cette fonction, seul le nom de la table présent dans la requête SQL varie.

1. Déplacez la fonction `selectAll()` de `ModelUtilisateur.php` vers `Model.php`.
2. Créez dans `ModelUtilisateur.php` une variable `$table` qui est `protected` (accessible uniquement dans la classe courante et ses classes filles) et `static` (qui ne dépend que de la classe, pas des objets).
3. Utilisez cette variable dans la fonction `selectAll()` de `Model.php` pour faire la requête sur la bonne table. Pour cela, accéder à la variable `$table` avec `static::$table` dans `Model.php`.

   Plus d'explications : La syntaxe `static::$table` est quelque peu subtile. Dans notre cas, elle permet que lorsque l'on appelle `ModelUtilisateur::selectAll()`, qui est héritée de `Model::selectAll()`, la variable `static::$table` aille chercher `ModelUtilisateur::$table` et non pas `Model::$table`.

4. Testez que votre site marche toujours.

Q 13. Passons à la fonction `select()`. Dans cette fonction, le nom de la table et la condition `WHERE` varie.

1. Déplacez la fonction `select()` de `ModelUtilisateur.php` vers `Model.php`.
2. Utilisez la variable statique `$table` de `ModelUtilisateur.php` pour remplacer le nom de la table.
3. Créez une variable statique `$primary` dans `ModelUtilisateur.php` qui contiendra le nom du champ de la clé primaire. Utilisez cette variable pour remplacer le nom de la clé primaire dans `select()`.

Q 14. Répétez la question précédente avec la fonction `delete()`.

Q 15. Passons à la fonction `update()`. Pour reconstituer la requête

```
'UPDATE utilisateur SET nom=:nom,prenom=:prenom,email=:email,login=:login WHERE login=:login',
```

il est nécessaire de pouvoir lister les champs de la table 'utilisateur'. Ces champs sont les entrées du tableau `$data` et c'est ainsi que nous allons les récupérer.

1. Déplacez la fonction `update()` de `ModelUtilisateur.php` vers `Model.php`.
2. Remplacer la table et le nom de la clé primaire par les variables adéquates.
3. Nous allons générer la partie `SET` à partir des clés du tableau associatif `$data`. Autrement dit, si `$data['un_champ']` existe, nous voulons rajouter la condition `'un_champ = :un_champ'` à `SET`.

   Indice : Utilisez la boucle `foreach ($tableau as $cle => $valeur)` pour récupérer les clés du tableau. Googler aussi la fonction `rtrim` de PHP qui pourra vous être utile pour enlever la virgule de trop à votre requête.

Q 16. Répétez la question précédente avec la fonction `insert()`.

4.2 Adaptation du contrôleur

Q 17. Couper l'adaptation du contrôleur en petit bouts testables. Il faut aussi adapter les vues au fur et à mesure. Finalement, il faut faire quelques remplacements dans VIEW_PATH, ModelUtilisateur et les vues (comme viewErrorUtilisateur) pour simplifier la tâche.