Lycée Sainte Ursule

S.L.A.M. 4 – Projet n° 1

–

Documentation technique

Auteurs : **ETTORI Matteo – HEUZÉ Maximilien – MARION Joffrey**

Version : **1**

Date : **18/12/2017**

Logiciel : **Eclipse Oxygen (Version Octobre 2017)**

Sommaire

Partie 1 : Les bases de données

I – La base de données

II – Le FrameWork et les autres outils utilisés

Partie 2 : Les avantages et les contraintes d'utilisation

I – Avantages de l'utilisation du FrameWork

II – Contraintes de l'utilisation du FrameWork

Partie 3 : L'affichage, l'API et le MVC de l'application

I – L'affichage de l'application dans un navigateur

II – L'API Google Maps

III – Le MVC de l'application

Partie 1 :

Architecture, Framework et autres outils

I – La base de données

Dans le cadre de ce premier projet « **Homepage** » développé en unité SLAM 4, nous avons utilisé le SGBDR (Système de Gestion de Bases de Données Relationnelles) en langage MySQL, afin d'organiser et architecturer les données de l'application.

Il a été nécessaire de créer la base de données, ses tables, ses données et ses relations à partir du langage MySQL, BDD nommé bdd_homepage_rev1 (que l'on peut retrouver à l'adresse http://slamwiki.kobject.net/_media/slam4/bdd_homepage_rev1.sql).

Dans une application MVC (Modèle Vue Contrôleur) comme celle-ci utilisant le FrameWork **PHP-MV-UI**, il faut savoir que les données de la base sont récupérables uniquement lorsque les classes « **models** » sont créées par le biais de la ligne de commande « **Micro** » soit l'exemple suivant créant le projet MVC « **homep** » utilisant la BDD « **homepage** » :

```
C:\WINDOWS\system32>cd /xampp7/htdocs

C:\xampp7\htdocs>Micro new homep -b=homepage -m -a -q=semantic
```

Cela fait, les données de la base sont facilement utilisables et récupérables en utilisant les méthodes de la classe DAO héritée sur chaque contrôleur de l'application. Par exemple :

1. Ligne initialisant une variable récupérant toutes données de la table « **Site** » par le biais du modèle éponyme (voir la méthode « **getAll** ») :

```
$sites=DAO::getAll("models\Site");
```

2. Ligne initialisant une variable toutes données de la table « Site » spécifique au site d'identifiant déclaré dans la variable « **$id** » par le biais du modèle éponyme (voir la méthode « **getOne** ») :

```
$site=DAO::getOne("models\Site", $id);
```

De cette façon, les données récupérées peuvent être utilisées par la suite, soit :

- Par l'affichage de tableau en les stockant dans un tableau « **dataTable** » ;
- Par l'affichage d'un formulaire « **dataForm** » en les stockant dans les différents champs ;
- Par l'affichage simple par l'appel des méthodes de classe « **Getteurs/Setteurs** » initialisées dans les classes correspondantes.

II – Le Framework et les autres outils utilisés

Au travers de cette application, nous avons utilisé :

- Le FrameWork **PHP-MV-UI** version 1.0.7 () et respecte tous les principes liés au MVC (Model View Controller – Modèle Vue Contrôleur) :
  - ✓ Lien d'installation : https://github.com/Mattori/phpMv-UI
  - ✓ Documentation : http://phpmv-ui.kobject.net/
- L'éditeur de code **Eclipse Oxygen** version Octobre 2017 :
  - ✓ Lien du site de téléchargement : https://www.eclipse.org/
  - ✓ Documentation : https://www.eclipse.org/documentation/
- Un serveur XAMPP sur lequel est hébergé :
  - ✓ L'application en cours de développement et/ou téléchargée depuis GitHub ;
  - ✓ La base de données de l'application ;
  - ✓ Tout le panel d'utilisation (code, sources…) du FrameWork.

Partie 2 :

Les avantages et les contraintes d'utilisation

I – Les avantages de l'utilisation du FrameWork

Avec le FrameWork **PHP-MV-UI**, il existe des avantages certains au niveau de l'utilisation :

- L'établissement d'un lien avec la BDD avec les « models » très pratique avec la commande « Micro » vu précédemment ;
- L'utilisation simple de la récupération et de l'utilisation des données par le biais des méthodes de la classe native « **DAO** » héritées par les contrôleurs ;
- Une utilisation simple des méthodes natives utilisables une fois le principe de fonctionnement du FrameWork compris ;
- Une visibilité de la BDD sous forme de MCD (Modèle de Conception des Données) pratique par l'utilisation de la plate-forme de vues des requêtes SQL PHPMyAdmin.

II – Les contraintes de l'utilisation du FrameWork

Bien que ce FrameWork contienne des avantages certaines, il existe également certaines contraintes au niveau de son utilisation :

- La quasi-obligation de faire le **lien avec la BDD** par le biais des « **models** » du MVC de l'application, faisable à la main mais pratiquement toujours par le biais de la commande « **Micro** » vu précédemment ;
- La gestion de la connexion utilisateur peu pratique si nous ne sommes pas munis d'une classe gérant la connexion des utilisateurs.

Partie 3 :

L'affichage, l'API et le MVC de l'application

I – L'affichage de l'application dans un navigateur

Comme toutes les applications utilisant le principe du MVC, toutes les pages sont liées à un contrôleur pour l'affichage, ce dernier affichant le contenu d'une page par le biais du chargement d'une vue HTML.

Si on prend l'exemple de la vue « **index.html** » permettant d'afficher le contenu de la page par le biais du contrôleur « **UserController** », nous utilisons des méthodes « **connexion** » et « **submit** » héritée de la classe « **ControllerBase** » pour pouvoir se connecter avec un utilisateur enregistré.

Ces méthodes sont héritées depuis « **ControllerBase** » afin d'éviter de recopier les fonctions de connexion sur chaque contrôleur, car il est possible de se connecter sur n'importe quel contrôleur de l'application.

Méthode « **connexion** » dans « **ControllerBase** » :

```php
public function connexion($ctrl,$action) {
    $frm=$this->jquery->semantic()->defaultLogin("connect");
    $frm->fieldAsSubmit("submit","green",$ctrl."/".$action,"body");
    $frm->removeField("Connexion");
    $frm->setCaption("login", "Identifiant");
    $frm->setCaption("password", "Mot de passe");
    $frm->setCaption("remember", "Se souvenir de moi");
    $frm->setCaption("forget", "Mot de passe oublié ?");
    $frm->setCaption("submit", "Connexion");
    echo $frm->asModal();
    $this->jquery->exec("$('#modal-connect').modal('show');",true);
    echo $this->jquery->compile($this->view);
}
```

Méthode « **submit** » dans « **ControllerBase** » :

```php
public function submit(){
    $id=RequestUtils::get('id');
    $user=DAO::getOne("models\Utilisateur", "login='".$_POST["login"]."'");
    if(isset($user)){
        $_SESSION["user"] = $user;
        $this->jquery->exec("$('body').attr('style','background: url(".$user->getFondEcran().")');",true);
    }
    $this->index();
}
```

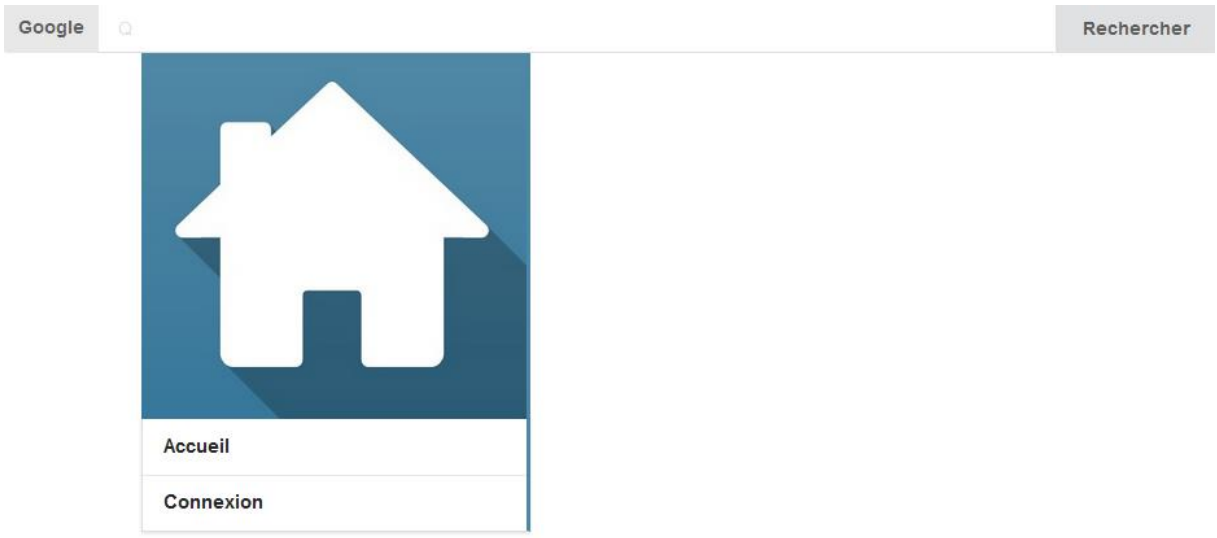
Utilisation dans la méthode « **index** » de « **UserController** » :

```
$menu->setPropertyValues("data-ajax", ["", "connexion/UserController/submit"]);
```

Il en est de même pour la méthode « **deconnexion** », qui dans le cas où l'utilisateur enregistré est déjà connecté, il est possible de se déconnecter par le bouton **Déconnexion**.

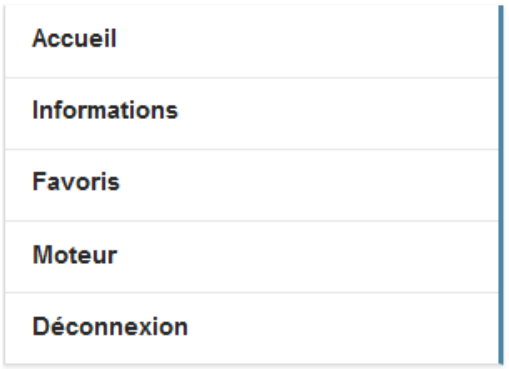
Rendu lors de la connexion :

- Avant le clic sur le bouton de connexion :



- Après le clic sur le bouton de connexion :

- Après le clic sur le bouton de validation du formulaire :



Hormis la connexion, toutes les méthodes influents directement sur les différentes propriétés des utilisateurs, du statut d'utilisateur normal à l'administrateur global, n'ont pas de particularité, sachant que tout se gère pratiquement depuis des méthodes affectant directement la BDD.

II – L'API Google Maps

Notre application incluant la géolocalisation de l'établissement du site, nous avons utilisé l'API Google Maps afin d'afficher l'emplacement exacte de l'établissement du site.

Pour ce faire, nous avons créé une méthode privée « **_generateMap** » récupérant en paramètre la latitude et la longitude indiquée dans les informations du site de l'établissement pour afficher la carte par la suite dans la vue « index.html » correspondante.

Nous pouvons prendre l'exemple avec la vue « **index.html** » de « **SiteController** » :

Méthode privée « **_generateMap** » dans « **SiteController** » :

```
private function _generateMap($lat,$long){
    return "
    <script>
        // DÃ©claration de la carte Google Maps
        var map={};

        // Fonction d'initialisation de la carte, de ses Ã©lÃ©ments et de ses Ã©vÃ¨nements
        function initMap() {
            // Options de la carte
            var optionsMap = {
                zoom: 17,
                center: {lat: {$lat}, lng: {$long}},
                mapTypeId: google.maps.MapTypeId.ROADMAP
            }
            // Affectation de la carte
            map = new google.maps.Map(document.getElementById('map'), optionsMap);

            // Options du cercle
            var optionsCercle = {
                map: map,
                center: map.getCenter(),
                radius: 50
            }
            // Affectation du cercle
            var cercle = new google.maps.Circle(optionsCercle);

            // Ajout d'un Ã©vÃ¨nement lorsque l'on clique sur la carte
            map.addListener('click',function(event){
                // Affectation de la valeur de la div d'id 'frmSite-latitude' Ã  la valeur de la latitude de l'Ã©vÃ¨nement
                document.getElementById('frmSite-latitude').value=event.latLng.lat();

                // Affectation de la valeur de la div d'id 'frmSite-latitude' Ã  la valeur de la longitude de l'Ã©vÃ¨nement
                document.getElementById('frmSite-longitude').value=event.latLng.lng();

                // Affectation de la valeur de la div d'id 'frmSite-latitude' Ã  la valeur de la longitude de l'Ã©vÃ¨nement
                document.getElementById('frmSite-longitude').value=event.latLng.lng();
            })

            // Ajout d'un Ã©vÃ¨nement lorsque l'on change la latitude de la div d'id 'frmSite-latitude'
            frmSite-latitude.addListener('change', function(event){
                // Affectation de la valeur de la cible de l'Ã©vÃ¨nement Ã  la valeur de la latitude de la carte
                event.target.value=map.latLng.lat();
            })
        }
    </script>
    <script src='https://maps.googleapis.com/maps/api/js?key=AIzaSyDxz9dHENw-b-1TlNXw88v3rWtKqCEb2HM&callback=initMap'></script>
    ";
}
```

Chargement dans la méthode privée « **_form** » de « **SiteController** » :

```
$this->loadView("sites\index.html",["jsMap"=>$this->_generateMap($lat,$long)]);
```

Rendu dans le formulaire de modification d'un site :

III – Le MVC de l'application

Avec **Eclipse Oxygen**, la structure du MVC de l'application est particulière. En effet, celle-ci possède plusieurs éléments permettant le chargement des fichiers et des dossiers de l'application. Nous avons donc :

- La configuration :
  - ✓ Gérer par le fichier « **config.php** » avec le chemin où se trouve l'application, la configuration de sa BDD, les espaces de noms, les librairies, le cache, etc…
  - ✓ Gérer en partie par le fichier « **.htaccess** » avec la protection du chemin où se trouve l'application.
- Les contrôleurs :
  - ✓ Gérer dans le répertoire « **controllers** » du répertoire « **app** » contenant tous les contrôleurs de l'application
- Les modèles :
  - ✓ Gérer dans le répertoire « **models** » du répertoire « **app** » contenant tous les modèles de l'application
- Les vues :
  - ✓ Gérer dans le répertoire « **views** » du répertoire « **app** » contenant tous les modèles de l'application :
    - Les vues principales créées de base « **vHeader.html** » et « **vFooter.html** » dans le répertoire « **main** » ;
    - Les vues créées pour les contrôleurs ayant tous pour nom « **index.html** » dans des dossiers ayant pour nom le nom du contrôleur concerné avec un « **s** » en plus

- Les ensembles d'éléments :
  - ✓ Gérer dans le répertoire « **assets** » contenant tous les autres éléments de traitement de l'application : les fichiers CSS, les fichiers JS, les images, etc…