Modèle MVC sous Samane MVC

Ngor SECK
PHP senior developer

ngorsecka@gmail.com

+221 77 433 97 16

Youtube/facebook/LinkedIn

10/03/2019 11:28 PM

version 1.9.2

ORM Doctrine

SamaneMVC est un framework PHP développé par Ngor SECK un ingénieur, développeur et formateur en développement d'applications. Il est full stack développer en PHP, JAVA et C# .NET. Son expérience dans ces langages surtout le JAVA l'a aidé à rendre le framework top et de haut niveau. Toute la complexité est déjà gérée pour vous.
L'objectif du framework est d'aider les développeurs à avoir un modèle simple d'utilisation, très structuré et respectant toutes les conventions du langage PHP. Toute la communauté web est invitée à l'utiliser et à faire des retours d'expériences dans le but de le perfectionner. Les entreprises sont aussi invitées à l'utiliser vu sa simplicité, ce qui permettra d'avoir un gain en temps de développement en argent sur les projets.
Le framework respecte le modèle de conception MVC qui est un modèle très connu et très utiliser dans les architecture d'applications modernes. Vous pouvez le télécharger directement à partir du github ou bien et mieux en utilisant composer. De nos jours, il est conseiller d'utiliser un outil de gestion de dépendances dans les applications modernes. Ainsi en PHP, composer est l'outil phare prisé par les développeurs. Vous pouvez trouver la commande de téléchargement composer dans le fichier README du github du framework.
Ce document va vous permettre d'avoir une prise en main rapide du framework. Avec des explication et des bouts de code qui facilitent la compréhension.
Le système de routing utilisé est très simple, juste le nom du contrôleur suivi d'une de ces méthodes. Au cas où la méthode aurait un paramètre, vous pourrez ajouter une valeur du paramètre.
Pour une bonne gestion des vues, le moteur de rendu de vue utilisé est Smarty. Ce moteur utilise des fichiers d'extension .html. Il est possible d'y insérer des structures conditionnelles comme if et des structures répétitives pour itérer sur des données.
Au niveau des modèles, depuis la première version nous avons utilisés la chaine PDO de PHP mais avec cette version 1.5 ORM, c'est Doctrine qui est utilisé pour la partie accès aux données comme vous allez le constater dans les extraits de code. Il est à noter que c'est la principale mise à jour.
Toute la logique du système qui permet au framework de fonctionner est dans un dossier libs à la racine de Samane. Ce système tourne au tour de quatre (4) fichier : View.lib.class.php, Model.lib.class.php, Controller.lib.class.php et Bootstrap.libs.class.php. Le paradigme de programmation utilisé est bien de la programmation orientée objet. Par convention, pour les classes que vous allez créer (contrôleurs et modèles), vous devez ajouter l'extension .class en l'extension finale .php.
La classe View.lib.class.php gère le moteur de template Smarty et le chargement dynamique des vues c'est à dire des fichiers html.

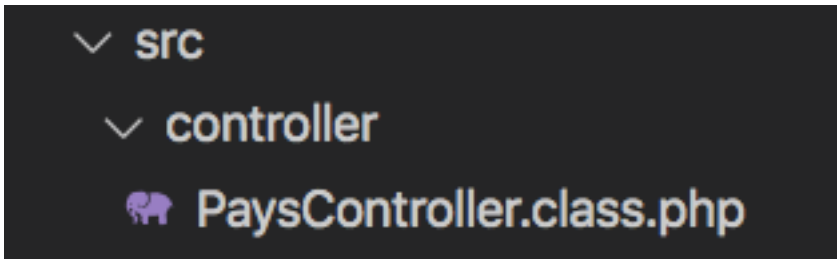
La classe Model.lib.class.php exploite le fichier de configuration des paramètres d'accès à une base de données ainsi que celui qui ouvre la connexion puis met à la disposition des toutes les modèles, qui seront créés, cette connexion.

La classe Controller.lib.class.php utilise une méthode de la classe View.lib.class.php pour le chargement des vues. Il met à la disposition de tous les autres contrôleurs du développeur cette méthode.
La classe Bootstrap.lib.class.php gère le routing c'est à dire l'exploitation des url saisies sur un navigateur, le chargement des contrôleurs ainsi que toutes les erreurs que le développeur pourrait commettre.
Dans la suite du document, nous allons vous expliquer le fonctionnement du framework pour les développeurs.
Pour être à l'aise avec le framework, vous devez comprendre la programmation orientées objet en PHP.

1. Gestion des Contrôleurs

1.a Création de contrôleur

Pour créer un contrôler, vous devez vous rendre dans le dossier **src** puis **contrôler** du framework. Et à ce niveau, créer un fichier PHP en respectant la convention de nommage des contrôleurs c'est à dire en ajoutant **.class** avant **.php**.



1.b Définition du code des controlleurs

1.b.1 Héritage

Chaque contrôleur que vous créez doit hériter de la super class Contrôler du framework. N'oublier d'effectuer l'héritage entre constructeur.

```
use libs\system\Controller;
```

```
class PaysController extends Controller
{
    public function __construct()
    {
        parent::__construct();
    }
```

1.b.2 Chargement des vues

Pour charger les views à partir des contrôleurs, c'est très simple. Utilise l'instruction PHP

```
return $this->view->load("pays/liste");
```

Ce code signifie que vous devez créer un dossier **pays** dans le dossier **view** qui se trouve dans le répertoire **src.** Puis créer un fichier **liste.html**
Voici le code du controlleur permettant le chargement de la vue *index.html*

```
public function liste()
{
   return $this->view->load("pays/liste");
}
```

1.b.3 Chargement des modèles

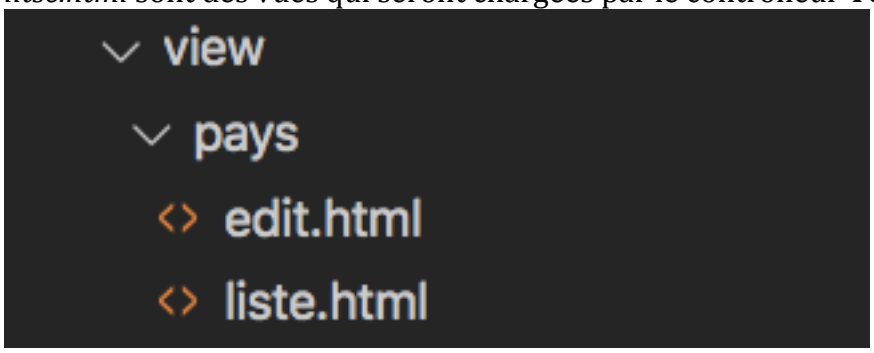
La couche modèle permet de récupérer ou de mettre à jour des données d'une base de données. Les modèles sont des classes qui héritent de la classe modèle du framework (*voir la partie modèle*).

Pour les chargements vous pouvez utiliser le mot clé **USE** si un namespace a été précisé au niveau du modèle. Puis vous n'avez qu'à instancier et commencer à utiliser votre modèle(trop simple non ☺).

2. Gestion des Vues

2.a Création

Les vues doivent être créées dans un répertoire de **view** qui se trouve dans le dossier **src.**
Voici un exemple : les fichiers *add.html, edit.html, get_id.html, index.html et litse.html* sont des vues qui seront chargées par le controlleur **Test**

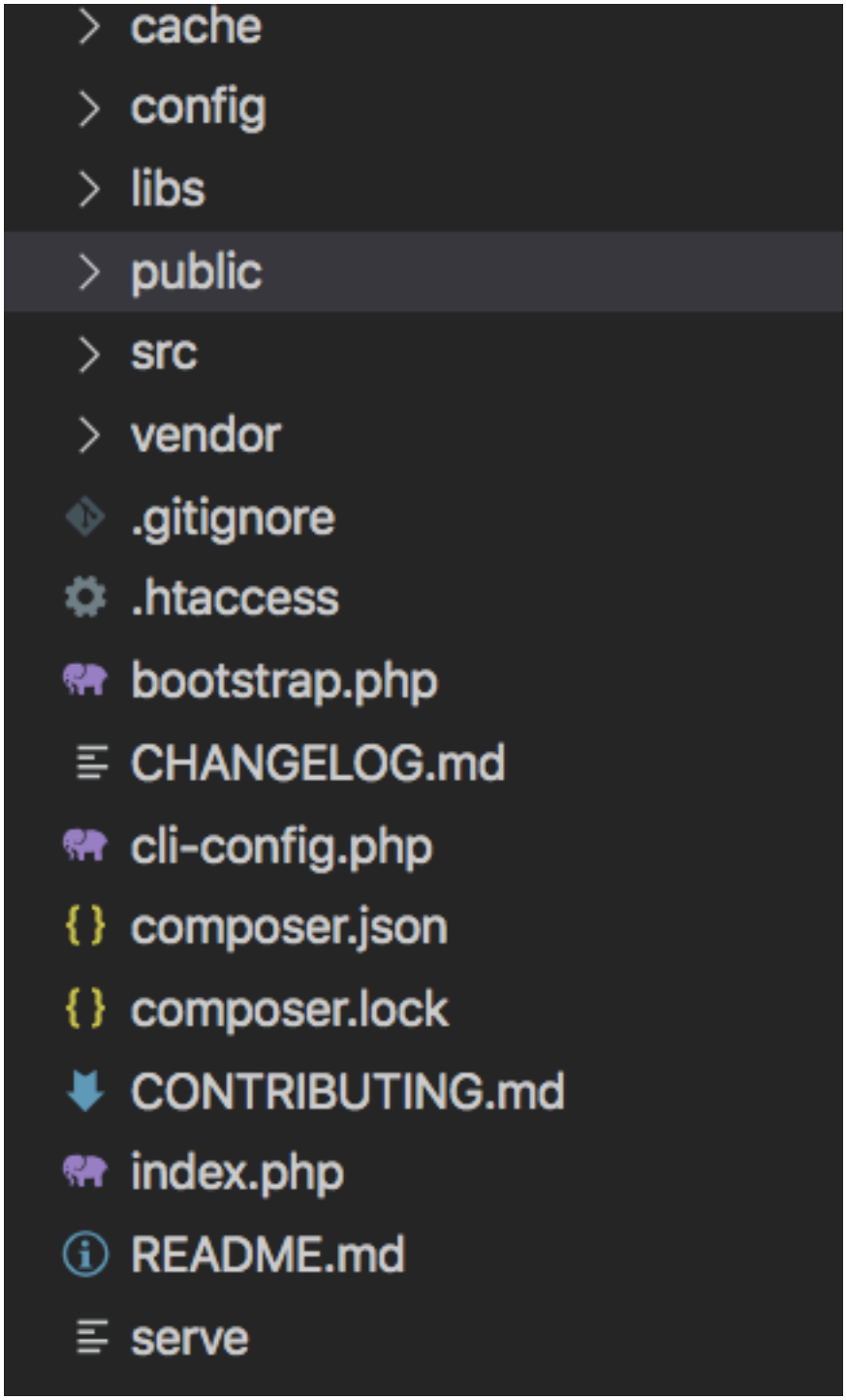


Pour la gestion des liens css ou js, vous pouvez vous inspirer des exemples de codes disponibles.

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>page get id</title>
        <!-- l'appel de {$url_base} vous permet de recupérer le chemin de votre site web  -->
        <link type="text/css" rel="stylesheet" href="{$url_base}public/css/bootstrap.min.css"/>
        <link type="text/css" rel="stylesheet" href="{$url_base}public/css/samane.css"/>
        <style>
            h1{
                color: #40007d;
            }
        </style>
    </head>
```

La variable **${url_base}** permet la récuperation de la racine de votre projet.
Exemples:
http://localhost/mesprojets/Samane_workspace/samanemvc/
oubien
http://localhost/samanemvc/
ou encore
http://www.samanemvc.sn/

Il suffit tout simplement de préciser cette instruction et le tour est joué. Tous vos fichiers CSS et JS ou bien même images doivent de localiser dans un dossier nommé **public** du framework.



2.b Définition du code des vues

Si une variable vient du contolleur, vous pouvez l'afficher directement dans la vue. Le moteur de template utilisé dans le framework est Smarty, donc vous devez respecter tout simplement les instructions du moteur.
Exemple : Nous allons afficher une valeur passée en paramètre dans une méthode de notre contrôleur Test au niveau d'une vue
*Voici le code **contrôleur** :*

```php
public function getID($id){
    $data['ID'] = $id;

    return $this->view->load("test/get_id", $data);
}
```

Et celui de la vue est :

```php
<h1>Valeur de l'identifiant revoyée par le controller : {$ID}</h1>
```

Trop simple non ☺

2.a Chargement des vues grâce aux modèles d'URL: les URL pattern

Ceci représente un modèle d'URL si le controller est Test.class.php

`localhost/mesprojets/Samane_workspace/samanemvc/Test/getID/1`

Et la partie la plus importante est

`Test/getID/1`

C'est à dire le nom du contrôler (**Test**) puis de la méthode (**getID**) et d'une valeur (**1**) dans le cas où la méthode en reçoit. Vous pouvez jeter un coup d'œil sur le code d'en haut.

Et voici votre rendu au niveau du navigateur.

Valeur de l'identifiant revoyée par le controller : 1

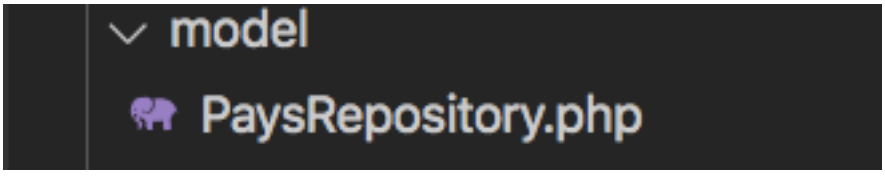
3. Gestion des Modèles

Comme préciser dans la partie 1.b.3, les modèles sont des classes d'accès aux données d'une base de données.

3.a Création

Les modèles doivent être créées dans un répertoire de **model** qui se trouve dans le dossier **src.**

Voici un exemple : *le model PaysRepository.php est créer dans le dossier model du framework.*



3.b Définition du code des modèles

3.b.1 Héritage

Chaque modèle que vous créez doit hériter de la super class Model du framework. N'oublier d'effectuer l'héritage entre constructeur.

```
namespace src\model;

use libs\system\Model;
```

Et pour l'héritage

```
class PaysRepository extends Model
{
    public function __construct()
    {
        parent::__construct();
    }
```

3.b.2 Chargement d'un modèle dans les controlleurs

Suivant l'espace de nom, utiliser le mot clé **USE**

```
use src\model\PaysRepository;
```

3.b.3 Exécution de requête SQL dans un modèle

Utiliser **$this->db** pour éxecuter toutes les requêtes SQL

```
/**
     * Methods with DQL (Doctrine Query Language)
     */
    public function getAll()
    {
        return $this->db->getRepository("Pays")->findAll();
    }
    public function get($id)
    {
        return $this->db->getRepository("Pays")->find(array("id"=>$id));
    }
    public function insert($pays)
    {
        $this->db->persist($pays);
        $this->db->flush();
    }
    public function update($pays)
    {
        //Le pays de la base de donnees
        $p = $this->get($pays->getId());
        /*Modification des donnees de ce pays avec les infos
        du pays passe en parametre cad $pays*/
        $p->setNom($pays->getNom());
```

```
        $p->setLatitude($pays->getLatitude());
        $p->setLongitude($pays->getLongitude());

        $this->db->flush();
    }

    public function delete($id)
    {
        $p = $this->get($id);

        $this->db->remove($p);
        $this->db->flush();
    }
```

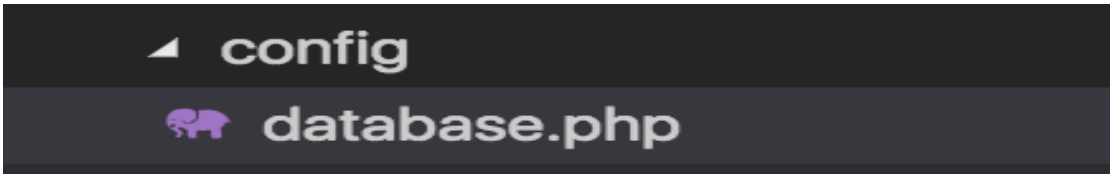
3.b.4 Utilisation d'un modèle dans les contrôleurs

Vous devez l'instancier

```
public function liste()
{
    $paysdb = new PaysRepository();
    $data["listePays"] = $paysdb->getAll();
    return $this->view->load("pays/liste", $data);
}
```

3.a Configuration de la connexion à une base de données

Vous devez vous rendre dans le fichier **database.php** qui se trouve dans dans le dossier **config** du framework.



Puis vous allez tout simplement effectuer des modifications en fonction de vos paramètres de connexion à une base de données. Voice le contenu de **config/database.php** :

```
/**
 * ORM
 */
$choix = "ORM";
/**
 * Turn to on or off your database
 */
```

```php
$etat = 'on'; //on or off

$orm = array(
       'dbname' => 'NomDeVotreBaseDeDonnees',//change your database name
       'user'     => 'root',
       'password' => '',
       'host' => '127.0.0.1',
       'driver' => 'pdo_mysql',
);
```

N'oublez pas que vous devez mettre à **on etat** pour que la configuration soit prise en compte.
Papez cette commande après création des entités sous Linux ou OSX
`vendor/bin/doctrine orm:schema-tool:update --force --dump-sql`
Papez cette commande après création des entités sous windows
`vendor\bin\doctrine orm:schema-tool:update --force --dump-sql`

Exemple complet

Le code du contrôleur **scr/controller/PaysController.class.php**

```php
<?php
use libs\system\Controller;

use src\model\PaysRepository;

class PaysController extends Controller
{
    public function __construct()
    {
        parent::__construct();
    }

    public function liste()
    {
        $paysdb = new PaysRepository();
        $data["listePays"] = $paysdb->getAll();
        return $this->view->load("pays/liste", $data);
    }

    public function insert()
    {
        extract($_POST);
        $pays = new Pays();
        $pays->setNom($nom);
        $pays->setLatitude($latitude);
        $pays->setLongitude($longitude);

        $paysdb = new PaysRepository();
```

```
        $paysdb->insert($pays);

        return $this->liste();
    }

    public function edit($id)
    {
        $paysdb = new PaysRepository();
        $data["pays"] = $paysdb->get($id);
        $data["listePays"] = $paysdb->getAll();
        return $this->view->load("pays/edit", $data);
    }
    public function update()
    {
        extract($_POST);
        $pays = new Pays();
        $pays->setId($id);
        $pays->setNom($nom);
        $pays->setLatitude($latitude);
        $pays->setLongitude($longitude);

        $paysdb = new PaysRepository();
        $paysdb->update($pays);

        return $this->liste();
    }

    public function delete($id)
    {
        $paysdb = new PaysRepository();
        $paysdb->delete($id);

        return $this->liste();
    }
}
?>
```

Le code des vues :
scr/view/pays/liste.html

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Welcome</title>
    <!-- l'appel de {$url_base} vous permet de recupérer le chemin de votre
site web  -->
    <link type="text/css" rel="stylesheet"
href="{$url_base}public/css/bootstrap.min.css"/>
```

```
</head>
<body>
<div class="nav navbar navbar-default navbar-fixed-top">
    <ul class="nav navbar-nav">
        <li><a href="{$url_base}">Accueil</a></li>
        <li><a href="{$url_base}Ville/liste">Ville </a></li>
        <li><a href="{$url_base}Pays/liste">Pays </a></li>
    </ul>
</div>
<div class="container" style="margin-top:80px;">
    <div class="col-md-6 col-xs-12" >
        <div class="panel panel-info">
            <div class="panel-heading">Liste des pays</div>
            <div class="panel-body">
                <table class="table table-bordered">
                    <tr>
                        <th>Identifiant</th>
                        <th>Nom</th>
                        <th>Latitude</th>
                        <th>Longitude</th>
                        <th>Action</th>
                        <th>Action</th>
                    </tr>
                    {foreach from=$listePays item=pays}
                        <tr>
                            <td>{$pays->getId()}</td>
                            <td>{$pays->getNom()}</td>
                            <td>{$pays->getLatitude()}</td>
                            <td>{$pays->getLongitude()}</td>
                            <td><a href="{$url_base}Pays/edit/{$pays-
>getId()}">Editer</a></td>
                            <td><a href="{$url_base}Pays/delete/{$pays-
>getId()}" onclick="return confirm('Voulez-vous supprimer
?');">Supprimer</a></td>
                        </tr>
                    {/foreach}
                </table>
            </div>
        </div>
    </div>
    <div class="col-md-6 col-xs-12">
        <div class="panel panel-info">
            <div class="panel-heading">Formulaire de gestion des pays</div>
            <div class="panel-body">
                <form method="post" action="{$url_base}Pays/insert">
                    <div class="form-group">
                        <label>Nom</label>
                        <input class="form-control" type="text" name="nom"/>
                    </div>
                    <div class="form-group">
                        <label>Latitude</label>
```

```
                        <input class="form-control" type="text"
name="latitude"/>
                    </div>
                    <div class="form-group">
                        <label>Longitude</label>
                        <input class="form-control" type="text"
name="longitude"/>
                    </div>
                    <div class="form-group">
                        <input class="btn btn-success" type="submit"
name="valider" value="Envoyer"/>
                        <input class="btn btn-danger" type="reset"
name="annuler" value="Annuler"/>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

scr/view/pays/edit.html

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Welcome</title>
    <!-- l'appel de {$url_base} vous permet de recupérer le chemin de votre
site web  -->
    <link type="text/css" rel="stylesheet"
href="{$url_base}public/css/bootstrap.min.css"/>
</head>
<body>
<div class="nav navbar navbar-default navbar-fixed-top">
    <ul class="nav navbar-nav">
        <li><a href="{$url_base}">Accueil</a></li>
        <li><a href="{$url_base}Ville/liste">Ville </a></li>
        <li><a href="{$url_base}Pays/liste">Pays </a></li>
    </ul>
</div>
<div class="container" style="margin-top:80px;">
    <div class="col-md-6 col-xs-12" >
        <div class="panel panel-info">
            <div class="panel-heading">Liste des pays</div>
            <div class="panel-body">
                <table class="table table-bordered">
                    <tr>
```

```
                        <th>Identifiant</th>
                        <th>Nom</th>
                        <th>Latitude</th>
                        <th>Longitude</th>
                        <th>Action</th>
                        <th>Action</th>
                    </tr>
                    {foreach from=$listePays item=pays}
                    <tr>
                        <td>{$pays->getId()}</td>
                        <td>{$pays->getNom()}</td>
                        <td>{$pays->getLatitude()}</td>
                        <td>{$pays->getLongitude()}</td>
                        <td><a href="{$url_base}Pays/edit/{$pays-
>getId()}">Editer</a></td>
                        <td><a href="{$url_base}Pays/delete/{$pays-
>getId()}" onclick="return confirm('Voulez-vous supprimer
?');">Supprimer</a></td>
                    </tr>
                    {/foreach}
                </table>
            </div>
        </div>
    </div>
    <div class="col-md-6 col-xs-12">
        <div class="panel panel-info">
            <div class="panel-heading">Formulaire de gestion des pays</div>
            <div class="panel-body">
                <form method="post" action="{$url_base}Pays/update">
                    <div class="form-group">
                        <label>Identifiant</label>
                        <input class="form-control" readonly type="text"
name="id" value="{$pays->getId()}"/>
                    </div>
                    <div class="form-group">
                        <label>Nom</label>
                        <input class="form-control" type="text" name="nom"
value="{$pays->getNom()}"/>
                    </div>
                    <div class="form-group">
                        <label>Latitude</label>
                        <input class="form-control" type="text"
name="latitude" value="{$pays->getLatitude()}"/>
                    </div>
                    <div class="form-group">
                        <label>Longitude</label>
                        <input class="form-control" type="text"
name="longitude" value="{$pays->getLongitude()}"/>
                    </div>
                    <div class="form-group">
```

```
                        <input class="btn btn-success" type="submit"
name="modifier" value="Modifier"/>
                        <input class="btn btn-danger" type="reset"
name="annuler" value="Annuler"/>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

Le code du model src/model/PaysRepository.php

```php
<?php
/*=================================================
MODELE MVC DEVELOPPE PAR Ngor SECK
samane@samanemvc.sn
(+221) 77 - 433 - 97 - 16
PERFECTIONNEZ CE MODELE ET FAITES MOI UN RETOUR
POUR TOUTE MODIFICATION VISANT A L'AMELIORER.
VOUS ETES LIBRE DE TOUTE UTILISATION.
=================================================*/
namespace src\model;

use libs\system\Model;

class PaysRepository extends Model
{
    public function __construct()
    {
        parent::__construct();
    }

    /**
     * Methods with DQL (Doctrine Query Language)
     */
    public function getAll()
    {
        return $this->db->getRepository("Pays")->findAll();
    }
    public function get($id)
    {
        return $this->db->getRepository("Pays")->find(array("id"=>$id));
    }
    public function insert($pays)
    {
        $this->db->persist($pays);
        $this->db->flush();
    }
    public function update($pays)
```

```
    {
        //Le pays de la base de donnees
        $p = $this->get($pays->getId());
        /*Modification des donnees de ce pays avec les infos
        du pays passe en parametre cad $pays*/
        $p->setNom($pays->getNom());
        $p->setLatitude($pays->getLatitude());
        $p->setLongitude($pays->getLongitude());

        $this->db->flush();
    }

    public function delete($id)
    {
        $p = $this->get($id);

        $this->db->remove($p);
        $this->db->flush();
    }
}
```

Le code de l'entité utilisée : une classe Pays dans src/entities

```
<?php
use Doctrine\ORM\Annotation as ORM;
use Doctrine\Common\Collections\ArrayCollection;
/**
 * @Entity
 * @Table(name="pays")
 **/
class Pays
{
    /** @Id @Column(type="integer") @GeneratedValue **/
    private $id;
    /** @Column(type="string") **/
    private $nom;
    /** @Column(type="decimal") **/
    private $longitude;
    /** @Column(type="decimal") **/
    private $latitude;
     /**
     * One lieu has many formations. This is the inverse side.
     * @OneToMany(targetEntity="Ville", mappedBy="pays")
     */
    private $villes;

    public function __construct()
    {
        $this->villes = new ArrayCollection();
    }
    public function getId()
    {
```

```
        return $this->id;
    }
    public function setId($id)
    {
        $this->id = $id;
    }

    public function getNom()
    {
        return $this->nom;
    }
    public function setNom($nom)
    {
        $this->nom = $nom;
    }

    public function getLongitude()
    {
        return $this->longitude;
    }
    public function setLongitude($longitude)
    {
        $this->longitude = $longitude;
    }

    public function getLatitude()
    {
        return $this->latitude;
    }
    public function setLatitude($latitude)
    {
        $this->latitude = $latitude;
    }

    public function getVilles()
    {
        return $this->villes;
    }
    public function setVilles($villes)
    {
        $this->villes = $villes;
    }
}

?>
```

Le code de l'entité utilisée : une classe Ville dans src/entities

```
<?php
use Doctrine\ORM\Annotation as ORM;

/**
```

```php
 * @Entity @Table(name="ville")
 **/
class Ville
{
    /** @Id @Column(type="integer") @GeneratedValue **/
    private $id;
    /** @Column(type="string") **/
    private $nom;
    /** @Column(type="decimal") **/
    private $latitude;
    /** @Column(type="decimal") **/
    private $longitude;
    /**
     * Many formation have one lieu. This is the owning side.
     * @ManyToOne(targetEntity="Pays", inversedBy="villes")
     * @JoinColumn(name="pays_id", referencedColumnName="id")
     */
    private $pays;

    public function __construct()
    {

    }
    public function getId()
    {
        return $this->id;
    }
    public function setId($id)
    {
        $this->id = $id;
    }

    public function getNom()
    {
        return $this->nom;
    }
    public function setNom($nom)
    {
        $this->nom = $nom;
    }

    public function getLatitude()
    {
        return $this->latitude;
    }
    public function setLatitude($latitude)
    {
        $this->latitude = $latitude;
    }

    public function getLongitude()
```

```
    {
        return $this->longitude;
    }
    public function setLongitude($longitude)
    {
        $this->longitude = $longitude;
    }

    public function getPays()
    {
        return $this->pays;
    }
    public function setPays($pays)
    {
        $this->pays = $pays;
    }
}

?>
```

Les URL
http://localhost/samanemvc/Pays/liste

http://localhost/samanemvc/Pays/edit/2