Doctrine

< Javascript >>

Doctrine est également un ORM qui peut être associé à CodeIgniter, il est beaucoup plus puissant, et plus complet.

- Site de référence Doctrine
- Documentation
- Téléchargement DoctrineORM-2.2.1-full

Installation

Doctrine est installé en tant que bibliothèque dans codeIgniter.
Créer un nouveau projet PHP, installer à nouveau CodeIgniter.
Avec l'archive Doctrine :

- Dézipper l'archive.
- Copier le dossier **Doctrine** de l'archive dans le dossier **application/libraries**.

Créer une classe Doctrine.php dans le dossier libraries :

|h application/libraries/Doctrine.php

```
<?php
class Doctrine
{
    // the Doctrine entity manager
    public $em = null;

    public function __construct()
    {
        // include our CodeIgniter application's database configuration
        require APPPATH.'config/database.php';
        // include Doctrine's fancy ClassLoader class
        require_once APPPATH.'libraries/Doctrine/Common/ClassLoader.php';

        // load the Doctrine classes
        $doctrineClassLoader = new \Doctrine\Common\ClassLoader('Doctrine',
APPPATH.'libraries');
        $doctrineClassLoader->register();

        // load Symfony2 helpers
        // Don't be alarmed, this is necessary for YAML mapping files
        $symfonyClassLoader = new \Doctrine\Common\ClassLoader('Symfony',
APPPATH.'libraries/Doctrine');
        $symfonyClassLoader->register();

        // load the entities
        $entityClassLoader = new \Doctrine\Common\ClassLoader('Entities',
APPPATH.'models');
```

```
        $entityClassLoader->register();

        // load the proxy entities
        $proxyClassLoader = new \Doctrine\Common\ClassLoader('Proxies',
APPPATH.'models');
        $proxyClassLoader->register();

        // set up the configuration
        $config = new \Doctrine\ORM\Configuration;

        if(ENVIRONMENT == 'development')
            // set up simple array caching for development mode
            $cache = new \Doctrine\Common\Cache\ArrayCache;
        else
            // set up caching with APC for production mode
            $cache = new \Doctrine\Common\Cache\ApcCache;
        $config->setMetadataCacheImpl($cache);
        $config->setQueryCacheImpl($cache);

        // set up proxy configuration
        $config->setProxyDir(APPPATH.'models/Proxies');
        $config->setProxyNamespace('Proxies');

        // auto-generate proxy classes if we are in development mode
        $config->setAutoGenerateProxyClasses(ENVIRONMENT == 'development');

        // set up annotation driver
        //$yamlDriver = new
\Doctrine\ORM\Mapping\Driver\YamlDriver(APPPATH.'models/Mappings');
        $driverImpl = $config->newDefaultAnnotationDriver(APPPATH.'models');
        $config->setMetadataDriverImpl($driverImpl);

        // Database connection information
        $connectionOptions = array(
            'driver' => 'pdo_mysql',
            'user' => $db['default']['username'],
            'password' => $db['default']['password'],
            'host' => $db['default']['hostname'],
            'dbname' => $db['default']['database']
        );

        // create the EntityManager
        $em = \Doctrine\ORM\EntityManager::create($connectionOptions,
$config);

        // store it as a member, for use in our CodeIgniter controllers.
        $this->em = $em;
    }
}
?>
```

Doctrine doit être ensuite chargé automatiquement avec autoload.php :

```
$autoload['libraries'] = array('database','doctrine');
```

Le chargement de la bibliothèque **database** est indispensable

Logiquement, Doctrine est prêt à fonctionner.
Vérifier que la page d'accueil ne produit pas d'erreurs : http://localhost/doctrine_CI/

Création des classes métier

Une classe métier correspond à la notion d'**entity** dans Doctrine.

Considérons la base de données suivante :

La base de données sera composée de 2 entities: utilisateur et categorie. La relation de type CIF entre utilisateurs et categories peut s'exprimer de la façon suivante :

- Chaque utilisateur appartient à 1 catégorie (manyToOne)
- Dans chaque categorie, on peut compter de 0 à n utilisateurs (oneToMany)

Le model utilisateur

Dans le dossier **application/models** :

- créer le fichier **utilisateur.php**
- générer ensuite les accesseurs sur les membres
- créer un constructeur sans paramètres

|h application/models/utilisateur.php

```php
<?php
/**
 * @Entity
 * @Table(name="utilisateurs")
 */
class Utilisateur {
    /**
     * @Id @Column(type="integer")
     * @GeneratedValue
     */
    private $id;
    /**
     * @Column(type="string")
     * @var string
     */
    private $nom;
    /**
     * @Column(type="string")
     * @var string
     */
```

```
    private $prenom;
    /**
     * @Column(type="integer")
     * @var integer
     */
    private $age;
    /**
     * @Column(type="boolean")
     * @var string
     */
    private $adulte;
    /**
     * @ManyToOne(targetEntity="Categorie")
     * @JoinColumn(name="categorie_id", referencedColumnName="id")
     */
    private $categorie;
}
?>
```

- Un **model** est un fichier contenant une classe dont le nom commence par une majuscule.
- Le nom du fichier doit être le même que celui de la classe, mais en minuscule.
- Le fichier doit être enregistré dans le dossier **application/models/**
- Doctrine peut utiliser plusieurs systèmes pour définir les modèles :
  - avec annotations dans le code php comme dans l'exemple
  - avec fichiers xml
  - avec fichiers yaml

Le model categorie

Dans le dossier **application/models** :

- créer le fichier **categorie.php**
- générer ensuite les accesseurs sur les membres
- créer un constructeur sans paramètres

|h application/models/categorie.php

```
<?php
/**
 * @Entity
 * @Table(name="categories")
 */
class Categorie{
    /**
    * @Id @Column(type="integer")
    * @GeneratedValue
    */
    private $id;
    /**
     * @Column(type="string")
     * @var string
     */
```

```
    private $nom;

    /**
     * @OneToMany(targetEntity="Utilisateur",mappedBy="categorie")
     */
    private $utilisateurs;
}
?>
```

Chargement des models

Le chargement peut être automatique, par le biais de application/config/autoload.php

```
$autoload['model'] = array('categorie','utilisateur');
```

ou bien se faire dans un contrôleur :

```
$this->load->model('utilisateur');
```

Gestion des utilisateurs

Contrôleur utilisateurs

Ajouter un contrôleur utilisateurs dans controllers :

- la méthode **all** charge tous les utilisateurs, et leur catégorie correspondante.
- Elle appelle ensuite la vue **v_utilisateurs** et lui passe les utilisateurs chargés (**users**)

|h application/controllers/utilisateurs.php

```
<?php
class Utilisateurs extends CI_Controller{
    public function all(){
        $query = $this->doctrine->em->createQuery("SELECT u FROM Utilisateur u
join u.categorie c where c.nom='Admin'");
        $users = $query->getResult();
        $this->load->view('v_utilisateurs',array('utilisateurs'=>$users));
    }
}
?>
```

Vues

Liste des utilisateurs

Créer la vue v_utilisateurs pour afficher la liste des utilisateurs : La variable $utilisateurs est récupérée par la méthode all du contrôleur utilisateurs

|h application/views/v_utilisateurs.php

```
<?php
foreach ($utilisateurs as $user){
    echo($user->nom." (".$user->cat->nom.")<br>");
}
?>
```

Tester en allant à l'adresse http://localhost/testPhp/utilisateurs/all/

Ajout d'utilisateur

Modification du contrôleur

Modifier le contrôleur utilisateurs :

- La méthode **add** permet d'afficher un formulaire **v_utilisateur_add** permettant d'ajouter un utilisateur en saisissant son nom.
- La méthode **submit_add** effectue la validation du formulaire en cas de succès de la validation puis appelle la vue **v_success_add**

|h application/controllers/utilisateurs.php

```
<?php
class Utilisateurs extends CI_Controller{
    public function add(){
        $this->load->helper(array('form', 'url'));

        $this->load->library('form_validation');

        $this->form_validation->set_rules('username', 'Username',
'trim|required|min_length[5]|max_length[12]|xss_clean');
        if ($this->form_validation->run() == FALSE)
        {
            $this->load->view('v_utilisateur_add');
        }
        else
        {
            $this->submit_add($_POST["username"]);
```

```
        }
    }

    public function submit_add($name){
        $user = new Utilisateur();
        $user->setNom($name);
        $this->doctrine->em->persist($user);
        $this->doctrine->em->flush();
    }

    public function all(){
        $query = $this->doctrine->em->createQuery("SELECT u FROM Utilisateur u
join u.categorie c where c.nom='Admin'");
        $users = $query->getResult();
        $nomCat="";
        foreach ($users as $user){
            if($user->getCategorie()!=null)
                $nomCat=$user->getCategorie()->getNom();
            echo($user->getNom()." ".$nomCat."<br>");
        }
    }
}
?>
```

Ajout des vues

La vue **v_utilisateur_add** sera appelée par l'intermédiaire du contrôleur **utilisateurs/add**

|h application/views/v_utilisateur_add.php

```
<html>
<head>
<title>Ajout utilisateur</title>
</head>
<body>

<?php echo validation_errors(); ?>

<?php echo form_open('utilisateurs/add/'); ?>

<h5>Nom d'utilisateur</h5>
<input type="text" name="username" value="<?php echo set_value('username');
?>" size="50" />

<div><input type="submit" value="Ajouter utilisateur" /></div>

</form>

</body>
</html>
```

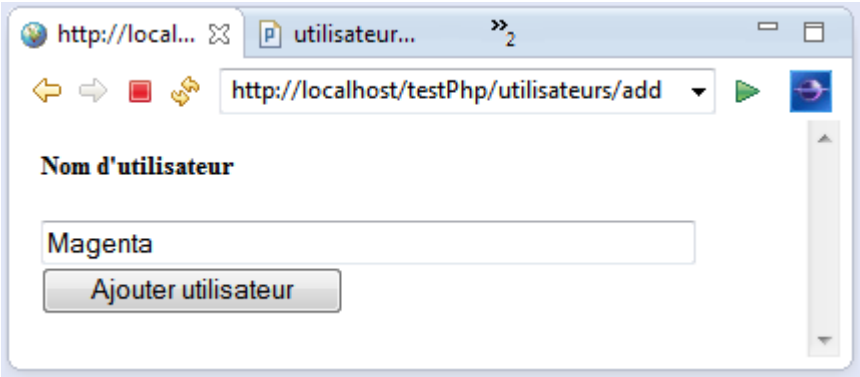
La vue **v_success_add** sera appelée après soumission du formulaire par le contrôleur

utilisateurs/submit_add

|h application/views/v_success_add.php

```
<?php
echo($user->nom." ajouté");
?>
```

Tester en allant à l'adresse : http://localhost/testPhp/utilisateurs/add/



Vérifier l'insertion dans la base de données du nouvel utilisateur.

Sur le même principe que pour les utilisateurs, en respectant MVC :

- Créer un contrôleur categories
- Afficher la liste des catégories, et les utilisateurs correspondants
- Créer la fonctionnalité d'ajout de catégorie
- Créer la fonctionnalité de modification d'une catégorie existante
- Créer la fonctionnalité de suppression d'une catégorie