# Exposé de Projet BD

Thème:

Manipulation et Connexion à une Base de données en PHP en utilisant PDO et les Frameworks Zend Framework et CodeIgniter

# Table des matières

# Introduction

**PHP** (***Hypertext Preprocessor***) est un langage de scripts libre et orienté objet depuis sa version 5, principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP. L'une des fonctionnalités que offre le PHP et qui fait l'objet de notre exposé aujourd'hui est la connexion aux Bases de Données ici **MySql**, par **PHP** puis via des **Framework** comme **Zend** et **CodeIgniter**.

# Un Peu d'histoire sur les framework

En France, les principaux autres Framework que l'on trouve sur le marché des applications professionnelles sont les suivants :

• Symfony : un projet mûr qui propose une architecture solide, mais légèrement plus rigide. Il est appuyé par une grande communauté d'utilisateurs ainsi qu'une entreprise (Sensio).

• Prado : un framework sérieux qui propose une architecture intéressante et un fonctionnement

Très spécifique.

• Copix : un projet mûr à destination du monde professionnel, qui est capable de répondre à de

nombreux besoins.

• Jelix : un framework français, comme Copix, de bonne qualité.

• CodeIgniter : un framework de plus en plus populaire pour sa simplicité et ses performances.

La souplesse de Zend Framework est telle que, quelle que soit la base choisie, une collaboration cohérente peut être mise en place avec d'autres Framework ou composants.

# Zend Framework

## Téléchargement du paquet Zend Compressé.

Télécharger la dernière version du paquet Zend à l'url suivante : http://framework.zend.com c'est l'adresse du site officiel.

Créer un dossier qu'on appellera Library dans C:/wamp/

Décompresser le fichier télécharger et copier le fichier « library » de zend framework dans le dossier C:/www/Library

## Configuration du serveur apache

Aller dans le fichier php.ini et modifier la directive include_path

**include_path = ".;C:/www/Library/library"**

Redémarrez apache et le tour est joué, le framework zend est installé et configuré.

# Les SGBD utilisables par Zend Framework

Zend Framework propose le support des SGBD via PDO.

PDO : PHP Data Object. PDO est disponible dans toutes les versions de PHP acceptées par Zend Framework. Il s'agit d'un socle commun d'accès aux SGBD, orienté objet.

Les différents SGBD au quelles on peut se connecter avec Zend Framework sont :

MySql, Microsoft SQL Server, Oracle, PostgreSQL, SQLite, IBM DB2 et Informix Dynamic Server (IDS)

# Manipulation de la base de données avec Zend Framework :

### *Connexion à une Base de données avec Zend Framework*

Avant de manipuler une base de données il faut d'abord avoir une connexion à la base de donnée. Pour créer une connexion à une base de données on utilise le code suivant :

```
<?php
$db = new Zend_Db_Adapter_Pdo_Mysql(array(
    'host '     => '127.0.0.1',
    'username' => 'root',
    'password' => '',
    'dbname'    => 'mysql'
));
```

La classe Zend_DB propose une méthode statique **factory()** qui peut faire exactement la même chose. Elle apporte cependant plus de flexibilité si l'on souhaite changer de SGBD dans le futur.

### *Envoyer une requête à la database et récupérer*

Pour envoyer une requête on utilise la méthode ***fetch*** chargée d'envoyer une requête et d'en récupérer les résultats.

Mais pour plus de précision on peut utiliser les différentes sous méthodes fetch qui sont :

| Sous méthodes fetch | Action effectué |
|---|---|
| fetchAll() | Récupère tous les résultats. |
| fetchRow() | Récupère le premier jeu de résultats. |
| fetchAssoc() | Récupère tous les résultats dans un tableau associatif |
| fetchCol() | Récupère tous les résultats, mais uniquement la première colonne demandée. |
| fetchOne() = fetchRow()+fetchCol() | Retourne la première colonne du premier jeu de résultats. |
| fetchPairs() | Récupère le résultat sous forme de tableau associatif. La conne 1 est en index, la colonne 2 en résultat |

### Exemple de récupération de résultat suite à une requête SELECT

```
< ?php // Inclusion du composant Zend_Loader
include 'Zend/Db/Adapter/Pdo/Mysql.php';
include 'Zend/Debug.php';
$db = new Zend_Db_Adapter_Pdo_Mysql(array(
    'host'=> '127.0.0.1',
    'username'=> 'root',
    'password'=> '',
    'dbname'=>'zendframework'
));
$query = "SELECT * FROM user";
```

```
$result = $db->fetchAll($query);
Zend_Debug::dump($result);
```

Toutes les méthodes **fetch** prennent un paramètre de bind : il s'agit d'une chaine ou d'un tableau de chaînes à remplacer lors de la requête. Zend Framework utilise aussi les requêtes préparées en permanence.

### Exemple des user présent dans la BD à partir de leur id

```
$query = "SELECT *
        FROM user
        WHERE id=:id";
$tableau_id = range(1, 4);
foreach ($tableau_id as $id) {
   $binds = array('id'=>$id);
   $result = $db->fetchRow($query, $binds);
   echo "Nom = ".$result['nom'] . "  Prenom = " . $result['prenom']. "  matricule = ". $result['matricule'] . "  id = ". $result['id'];
            echo "</br>";
            echo "</br>";
}
```

### Insertion d'un élément dans une table de la base de données.

```
try {
   $data = array('nom' => 'nom a insere', 'prenom' => 'prenom a insere','matricule' => '12P200', 'id'=>'5');
   $count = $db->insert("user", $data);
            //Le count que vous avez ici est le nombre de lignes insérées dans la table user.
   echo $count . " user insere";
            }
catch (Zend_Db_Exception $e) {
   printf("erreur de requête : %s", $e->getMessage());
            }
```

### Mise à jour de donnée dans la base de données

```
$updated = $db->update("user", array('nom' => "yoba update"), 'id=1');

echo $updated . " enregistrement(s) affecté";
```

### Suppression d'une ligne dans la Base de données

```
$conditions = array("id=5");

$deleted    = $db->delete("user", $conditions);

echo $deleted . " enregistrement(s) supprimé(s)";
```

### *Utiliser les passerelles vers les tables*

L'utilisation des passerelles vers les tables se fait par la classe Zend_Db_Table. En effet cette table permet de créer une passerelle entre une table de la base de données et une classe PHP. Chaque table peut être représentée par une classe et les méthodes proposées sur les objets d'instances de cette classe vont offrir de manière simple d'effectuer des opérations sur la table en question.

Il sera question en fait de faire une sorte de mapping de la table. Il est primordial de préciser lors du maping la clé primaire de la table. Dans le cas où la clé primaire est constituée de plusieurs colonnes il faut spécifier sous forme de tableau.

Une fois la table mappée elle aura besoin d'un adaptateur. Pour pouvoir faire ses connexions à la base de données.

## Le Framework CodeIgniter :

Il a été conçu dans le but de ne fournir que le strict minimum. Tout le reste est entièrement optionnel (même les bibliothèques gérant les bases de données et les sessions le sont).

Cela lui permet donc d'être doublement rapide. D'une part, sur le temps d'apprentissage du framework : vous verrez que vous vous sentirez très vite à l'aise. D'autre part, sur le temps de génération de votre page. Autant vous le dire tout de suite, CodeIgniter se retrouve souvent en très bonne position dans les benchmarks (je ne pourrai pas vous en dire plus sur ce point, au risque de ne plus être objectif).
Si l'on devait résumer le framework en une phrase, on dirait que CodeIgniter est une base réduite en fonctionnalités mais hautement performante, pouvant faire appel à des classes et à des fonctions très complètes lorsque le besoin s'en fait sentir.

### Le contrôleur :

Le contrôleur est ce qui va être appelé en premier. Il n'y a qu'un seul contrôleur par URL. Donc, deux URL différentes appelleront deux contrôleurs différents. Le contrôleur se charge d'appeler tous les composants : bibliothèques, helpers, vues, modèles, mais aussi de faire les nombreuses vérifications nécessaires.

Voici quelques exemples de fonctionnalités que peut fournir un contrôleur.

- Vérifier si le formulaire a bien été rempli.
- Vérifier si le visiteur a bien le niveau requis pour voir la page (administrateur par exemple).
- Changer le design selon les préférences du visiteur (via un cookie, par exemple).
- Contrôler les tokens (ou jetons) de sécurité. Si vous ne connaissez pas ces petites bêtes, ce n'est pas grave (tout du moins pour ce tutoriel).

Le contrôleur sera donc la partie la plus importante pour le développeur PHP. Tout ce qui se passe dans le contrôleur sera invisible aux yeux des visiteurs. Le contrôleur n'a pas pour rôle d'envoyer au navigateur un message d'erreur, il a pour rôle de trouver cette erreur. L'affichage de cette erreur se fera via une autre couche d'abstraction.

Avec CodeIgniter , vous verrez que chaque contrôleur est une classe qui est appelée selon l'URL. Et comme toute classe qui se respecte, elle possède un certain nombre de méthodes qui représenteront les différentes actions que le contrôleur peut effectuer .

Par exemple, si on a cette URL : **http://nomdedomaine.tld/news/voir_news.html**, cela signifie que CodeIgniter doit appeler dans la classe **News** la méthode **voir_news**. C'est donc l'URL qui appelle le contrôleur.

#### Notre première page

**Premier contrôleur**

```php
<?php

class Forum extends CI_Controller
{
 public function accueil()
 {
  echo 'Hello World!';
 }
}
```

Dans CodeIgniter , un contrôleur est un fichier contenant uniquement une classe. Tout se passera donc dans cette classe.

Écrivez cette classe dans un fichier forum.php et placez celui-ci dans le dossier ./application/controllers/. Si vous vous rendez à l'adresse **http://localhost/codeIgniter/index.php/forum/accueil/**, vous verrez la page « **Hello World!** »

### Création d'autres pages

```php
<?php

class Forum extends CI_Controller
{
 public function accueil()
 {
  echo 'Hello World!';
 }

 public function bonjour()
 {
  echo 'Salut à tous !';
 }

 public function manger()
 {
  echo 'Bon appétit !';
 }
}
```

### *La méthode Index*

Cette méthode a la particularité d'être appelée dans le cas où votre URL ne spécifie pas de nom de méthode.

```php
<?php

class Forum extends CI_Controller
{
 public function index()
 {
  echo 'Index';
 }

 public function accueil()
 {
  echo 'Hello World!';
 }
}
```

N'oublions pas que nous sommes dans une classe.

```php
<?php


class Forum extends CI_Controller
{
 public function index()
 {
  $this->accueil();
 }

 public function accueil()
 {
  echo 'Hello World!';
 }
}
```

*La méthode _remap*

_remap est une méthode très puissante. Elle vous permet de modifier la méthode que vous allez utiliser. Autrement dit, faire une redirection interne un peu plus dans le style de CodeIgniter.

```php
<?php

class Home extends CI_Controller
{
 public function accueil()
 {
  echo 'Bonjour';
 }

 public function maintenance()
 {
  echo "Désolé, c'est la maintenance.";
 }

 public function _remap($method)
 {
  $this->maintenance();
 }
}
```

## Les vues :

Les vues sont directement responsables de tout ce que l'on va envoyer aux navigateurs . Les vues seront donc composées principalement de HTML. Nous y trouverons aussi du PHP qui nous servira pour afficher des variables et faire quelques boucles.

Il n'y a pas de limite au nombre de vues. Alors qu'avec les contrôleurs, nous avions une URL par contrôleur, ici, nous pouvons très bien avoir cinq vues par contrôleur. Le rôle du contrôleur est aussi d'appeler la bonne vue. De plus, rien n'empêche le contrôleur de faire appel à plusieurs vues.

Par exemple, pour la page d'accueil de l'administration d'un site, nous pourrons trouver ce genre de vues :

- une vue contenant le formulaire de connexion à la zone administrateur ;
- une vue contenant un message d'erreur ;
- une vue contenant la page d'administration.

Le rôle de la vue s'arrête là. C'est au contrôleur de savoir quelle est la vue qu'il doit charger.

### vue.php

```php
<h1>Bonjour</h1>

<p>
    Ceci est mon paragraphe !
</p>

<p>
    Votre pseudo est <?php echo $pseudo; ?>.
</p>

<p>
    Votre email est <?php echo $email; ?>.
</p>

<p>
<?php if($en_ligne) : ?>
    Vous êtes en ligne.
<?php else: ?>
    Vous n'êtes pas en ligne.
<?php endif; ?>
</p>
```

new.php

```php
<?php

class News extends CI_Controller
{
 public function index()
 {
  $this->accueil();
 }

 public function accueil()
 {
  $data = array();
  $data['pseudo'] = 'Arthur';
  $data['email'] = 'email@ndd.fr';
  $data['en_ligne'] = true;

  // Maintenant, les variables sont disponibles dans la vue
  $this->load->view('vue', $data);
 }
}
```

La bibliothèque Database

Pour charger la bibliothèque et vous connecter à la base de données, vous devez utiliser la méthode Database du loader.

```php
<?php

$this->load->database();
```

Exemple de requête SQL :

```php
<?php

$resultat = $this->db->select('id, email')
                     ->from('utilisateurs')
                     ->where('pseudo', 'ChuckNorris')
                     ->limit(1)
                     ->get()
                     ->result();
```

Cette requête se lit comme ceci :

**« Sélectionne-moi les colonnes 'id' et 'email' de la table 'utilisateurs' où le champ 'pseudo' vaut 'ChuckNorris' et arrête-toi dès que tu auras 1 résultat ».**

Lorsque nous aurons des requêtes beaucoup plus compliquées, il sera peut-être judicieux de revenir aux requêtes sous forme de chaînes de caractères. Voici un exemple :

*Avec la méthode query :*

```php
<?php

// Mise en place de notre requête
$sql = "SELECT `id`,
`email`
FROM `utilisateurs`
WHERE `pseudo` = ?
LIMIT 0,1
;";

// Les valeurs seront automatiquement échappées
$data = array('ChuckNorris');

// On lance la requête
$query = $this->db->query($sql, $data);

// On récupère le nombre de résultats
$nb_resultat = $query->num_rows();

// On parcourt l'ensemble des résultats
foreach($query->result() as $ligne)
{
 echo $ligne->id;
}

// On libère la mémoire de la requête (fortement conseillé pour
lancer une seconde requête)
$query->free_result();
```

## Les modèles :

Cette dernière couche d'abstraction permet de faire le lien entre le contrôleur et la base de données. Il est toujours conseillé de séparer convenablement les requêtes que vous envoyez à la base de données et les instructions qui vont utiliser ces données.

Le modèle permettra donc d'envoyer des requêtes à la base de données et d'en retourner le résultat sous forme brute (sans HTML). Nous y trouverons donc des fonctionnalités typiques des bases de données (ajouter , modifier , supprimer ...).

Dans la plupart des cas, si les bases de données ont bien été pensées, il y aura un modèle pour chaque table.

Prenons l'exemple d'une table sauvegardant des news. Le modèle associé à cette table devra être composé de ces fonctionnalités :

- ajouter ;
- modifier ;
- supprimer ;
- nombre_news ;
- liste_news ;
- news.

Eh bien, pour réaliser cela avec CodeIgniter , vous allez devoir créer une classe que nous appellerons News_model, et nous implémenterons toutes les méthodes dont nous aurons besoin pour manipuler les news dans la base de données. Il nous suffira alors d'exécuter les méthodes de ce modèle depuis le contrôleur. Pour en finir avec les modèles, il faut que vous sachiez qu'il est facultatif. Vous avez la possibilité de n'en inclure aucun. Ce n'est pas obligatoire de charger un modèle dans le cas où vous ne souhaitez pas exécuter de requêtes avec votre base de données.