

# **Systemes d'information**

## **Développement d'une application Web**

**Jean-François Berdjugin**  
**Vincent Lestideau**

---

# Systemes d'information: Développement d'une application Web

par Jean-François Berdjugin et Vincent Lestideau

Date de publication Septembre 2012

## Résumé

Nous allons construire, une application Web proposant la vente de tee-shirts. Notre application repose sur une architecture trois tiers<sup>1</sup> ; la couche présentation utilise XHTML (eXtensible HyperText Markup Language) ; la couche métier utilise PHP (Hypertext Preprocessor) et la couche d'accès aux données utilise un (SGBD) relationnel. Nous n'utiliserons pas dans cet exemple JScript (JavaScript) et AJAX (Asynchronous JavaScript and XML (Extensible Markup Language)) ce langage sera vu ce semestre en programmation (INF350).

Le développement repose sur le patron de conception MVC (Modèle Vue Contrôleur) et une approche fonctionnelle non objet pour les TD, l'accès aux bases de données sera réalisé en utilisant les méthodes de PDO (PHP Data Objects). En TP nous utiliserons le framework CodeIgniter et l'ORM (Object-Relational Mapping) Doctrine.

---

<sup>1</sup>Un tiers est une couche.

---

---

# Table des matières

1. Présentation du projet .....	1
1. Séances et évaluations .....	1
2. Présentation de l'architecture et des outils .....	2
1. Présentation .....	2
2. Première mise en oeuvre .....	3
2.1. Apache et eclipse .....	3
2.2. PHP .....	4
2.3. MySQL .....	4
3. Les TD, mon premier site Web .....	6
1. Présentation rapide du langage .....	6
1.1. Les types et variables .....	6
1.1.1. Présentation .....	6
1.2. Les constantes .....	9
1.3. Les structures de contrôles .....	10
1.3.1. Présentation .....	10
1.3.2. Les formes alternatives .....	12
1.3.3. Exercices .....	12
2. La communication entre le serveur Web et PHP .....	14
2.1. Exercices .....	14
2.1.1. Communication en POST .....	14
2.1.2. Communication en GET .....	15
3. Le MVC (Modèle Vue Contrôleur) .....	15
4. Le développement .....	17
4.1. Le contrôleur principal et les constantes .....	17
4.2. Le contrôleur de page utilisateur .....	18
4.3. La page d'accueil pour afficher l'ensemble des utilisateurs .....	19
4.3.1. Une courte introduction à PDO (PHP Data Objects) .....	20
4.3.2. Le modèle .....	20
4.3.3. Le contrôleur .....	21
4.3.4. La vue .....	21
4.4. L'ajout d'un utilisateur .....	22
4.4.1. Une courte introduction aux requêtes paramétrées .....	22
4.4.2. Le modèle .....	22
4.4.3. Le contrôleur .....	22
4.4.4. La vue .....	22
4.5. La suppression d'un utilisateur .....	23
4.5.1. Le modèle .....	23
4.5.2. Le contrôleur .....	23
4.5.3. La vue .....	23
4.6. La modification d'un utilisateur .....	23
4.6.1. La vue accueil_view.php .....	23
4.6.2. Le modèle .....	23
4.6.3. La méthode update du contrôleur .....	23
4.6.4. La vue update_view.php .....	23
4.6.5. La méthode updateOk du contrôleur .....	23
4. Les TP, utilisation d'un framework et d'un ORM .....	24
1. Mise en place .....	25
2. Développement .....	25
2.1. Les principaux fichiers de CodeIgniter .....	25
2.2. Notre modèle .....	26
2.2.1. Étude de notre modèle .....	27
2.2.2. Création de nos tables .....	28
2.2.3. Le peuplement de nos tables .....	28
2.3. Le frontoffice premier partie .....	28
2.3.1. Affichage de la liste des galeries (accueil_view) .....	28

2.3.2. Modification de la vue accueil_view .....	29
2.4. Le backoffice premier partie .....	29
2.5. Le frontoffice suite et fin .....	29
2.5.1. Galerie utilisateur .....	29
2.5.2. Afficher une image et sa description .....	30
2.6. Backoffice suite et fin .....	30
2.6.1. L'administration des données personnelles .....	31
2.6.2. L'administration des comptes .....	31
3. Devoir maison .....	32
A. Création des modèles .....	34
1. MySQL Workbench .....	34
2. PHPMyAdmin .....	38
3. CodeIgniter et Doctrine .....	39
3.1. Import du projet vide dans le workspace .....	39
3.2. Création de l'allias .....	40
3.3. Configuration de CodeIgniter .....	41
3.4. Le reverse Engineering .....	42
3.5. Création des dépôts .....	43
3.6. La modification des modèles .....	43
3.7. Génération de la base à partir du modèle .....	44
3.8. Amélioration des entités .....	44

---

## Liste des illustrations

2.1. Architecture .....	2
3.1. Diagramme de classe de la classe Utilisateur .....	14
3.2. Diagramme de classe de la classe GestionUtilisateur .....	14
3.3. Communication entre le serveur Web et PHP .....	14
3.4. MVC en PHP .....	16
3.5. MVC diagramme de séquence .....	17
3.6. Navigation .....	19
3.7. La page d'accueil .....	20
3.8. Le diagramme de classe de la classe GestionUtilisateurs .....	20
3.9. La page d'accueil sans ses données .....	21
4.1. Les modules de CodeIgniter .....	26
4.2. Code Igniter et Doctrine .....	27
A.1. Création d'un modèle Entité Relationnel .....	35
A.2. Le choix du nom et de l'encodage du schéma .....	36
A.3. Création de deux tables vides .....	36
A.4. La table utilisateur .....	36
A.5. La table image .....	36
A.6. La conception du diagramme .....	37
A.7. Création d'une association 1-n .....	37
A.8. Renommage des champs générés .....	37
A.9. Modification d'une contrainte d'intégrité référentielle .....	37
A.10. Un utilisateur et une base .....	39
A.11. Import du projet .....	40
A.12. Création de l'allias .....	41

---

## Liste des exemples

3.1. Variables .....	7
3.2. Tableaux .....	8
3.3. Classe Point .....	9
3.4. Définition et utilisation d'une constante .....	10
3.5. Exemple d'utilisation du if .....	10
3.6. Exemple d'utilisation du if else .....	10
3.7. Exemple d'utilisation du elseif .....	10
3.8. Exemple d'utilisation du switch .....	11
3.9. Exemple de boucle while .....	11
3.10. Exemple de boucle do while .....	11
3.11. Exemple de boucle for .....	11
3.12. Exemple de boucle foreach .....	12

---

# Chapitre 1. Présentation du projet

## 1. Séances et évaluations

Nous avons trois séances de TD (Travaux Dirigés) de trois heures où vous serez deux par machine. Les séances de TD seront suivies de quatre séances de TP (Travaux Pratiques) de trois heures ou vous serez un par machine. Nous aurons deux évaluations la première à la fin des séances de TP, sur machine et d'une durée de une heure trente ; la seconde sous la forme d'un devoir à la maison personnel dont le sujet doit être choisi avec l'enseignant à la fin des séances de TD.

Pour le devoir maison vous devez une fois le sujet validé par l'enseignant remettre :

- un pdf unique contenant : le sujet, le plan du site, les cas d'utilisation, le diagramme de classe du modèle et une notice de déploiement
- Le code source du site et les ressources.

Vous devrez en 10 minutes installer et présenter votre application Web devant un enseignant et un sujet écrit.

---

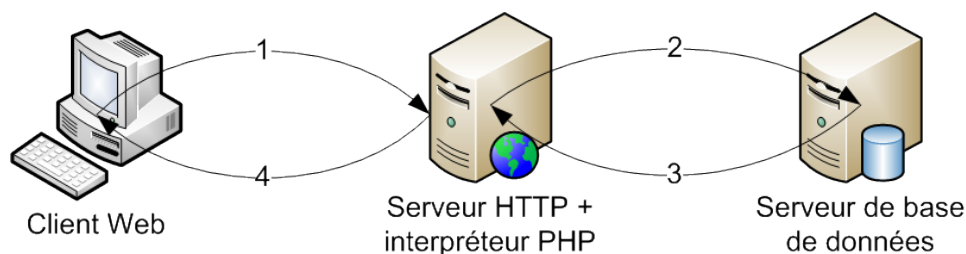
# Chapitre 2. Présentation de l'architecture et des outils

Le développement Web repose sur une architecture distribuée, nous n'avons plus un seul programme sur une seule machine mais un ensemble réparti sur plusieurs machines communicantes.

## 1. Présentation

Nous allons utiliser une architecture Web composée d'un serveur HTTP (HyperText Transfer Protocol), d'un SGBD (Système de Gestion de Base de Données), d'un client Web et enfin de PHP. PHP est un langage de script libre permettant de produire des pages Web dynamiques<sup>1</sup>. Nous avons un client Web, un serveur Web, une application PHP et un SGBD qui contient les bases de données.

Figure 2.1. Architecture



L'obtention d'une page dynamique suit les étapes suivantes :

1. Tout commence par le client Web qui émet une requête vers un serveur HTTP.
2. Le serveur analyse la requête HTTP et en fonction de la ressource demandée renvoie cette ressource ou soustrait la gestion de la requête à une autre application. Pour ce qui nous intéresse, toutes les ressources en `.php` sont envoyées vers l'interpréteur PHP celui-ci va gérer la réponse soit directement soit en faisant appel à un SGBD.
3. Le SGBD fournit les enregistrements à l'interpréteur PHP.
4. L'interpréteur PHP génère la page Web, la fournit au serveur HTTP et enfin le serveur HTTP renvoie la réponse au client.

Pour l'ensemble de nos serveurs nous avons choisi une solution intégrée : WAMP (Windows, Apache, MySQL, PHP). WAMP est disponible à l'URL (Uniform Resource Locator) suivante : <http://www.wampserver.com/>. Il existe d'autres solutions intégrées aussi nommées plate-forme de développement Web, comme easyphp, XAMPP (X Apache MySQL Perl PHP), LAMP (Linux, Apache, MySQL, PHP), ... .

### Important

Malgré tous les efforts pour rendre le code indépendant de l'architecture, il ne l'est jamais totalement, il vous faut donc lors de vos projets tutorés connaître votre hébergeur et sa configuration.

Chaque serveur dispose de ses fichiers de configuration, avec un hébergement mutualisé, vous ne pourrez y avoir accès. Voici les serveurs que le nous allons mettre en oeuvre.

#### Apache

Le serveur Web a pour principal fichier de configuration `httpd.conf`, vous le verrez en détail pendant les TP de réseaux. Nous ne l'utiliserons pas directement mais nous utiliserons l'interface "graphique" de WAMP et des fichiers `.htaccess`. Les fichiers `.htaccess` sont des fichiers de configuration des serveurs Web Apache

---

<sup>1</sup>PHP est un langage interprété, impératif et objet depuis la version 1.5. Il peut être utilisé en local sans serveur Web.



utilisables sans droits d'administration et qui permettent de redéfinir des directives de `httpd.conf`. Les fichiers `.htaccess` sont généralement le seul moyen de redéfinir une configuration d'apache offert par votre hébergeur.

## PHP

Le principal fichier de configuration est `php.ini`, ce fichier permet au travers de directives de définir le comportement de PHP en spécifiant les extensions, les répertoires les chemins d'accès aux fichiers, ... . Vous ne pourrez accéder au `php.ini` d'un fournisseur aussi certaines directives sont utilisables dans des fichiers `.htaccess` et `.user.ini`<sup>2</sup>.

## MySQL

Le principal fichier de configuration est `my.ini`, nous ne l'utiliserons pas et nous administrerons le SGBD en utilisant une application PHP : *phpMyAdmin*.

Nous utiliserons comme l'année précédente *eclipse* mais avec une version préconfigurée pour le développement Web : *Eclipse for PHP Developers* (<http://www.eclipse.org/>).

# 2. Première mise en oeuvre

Nous allons créer et afficher notre première page Web dynamique puis nous créerons une base de données avec une table et des enregistrements au sein du SGBD.

## 2.1. Apache et eclipse

Démarrer WAMP serveur (il lance l'ensemble des services) puis consulter l'URL : `http://localhost/`. Votre serveur apache fonctionne, mais comment pointer vers un répertoire de publication Web: en utilisant un alias.

1. Créer sous le disque "public" (d:) un répertoire nommé `workspace_inf340`, vous ne pouvez utiliser votre `z`: le serveur apache ne possède pas de droits sur ce dernier.
2. Lancer eclipse et choisir le répertoire précédent comme workspace.
3. Importer le nouveau projet PHP nommé `inf340_td`.
4. Dans le répertoire `exercices/exercice1`, vous trouverez un fichier nommé `test1.php`, qui contient le code suivant :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="fr" xml:lang="fr">
  <head>
    <title>XHTML 1.0 Strict Test</title>
  </head>
  <body>
    <p>
      <?php phpinfo();?>
    </p>
  </body>
</html>
```

5. Créer dans apache un alias nommé `inf340td` pointant vers le répertoire physique `d:\workspace_inf340\inf340_td`. Ainsi apache pourra accéder aux ressources de votre projet. *Dans la pratique, pour des raisons de sécurité, seul les scripts devant apparaître dans l'URL doivent être dans le répertoire de publication et donc directement accessibles par les clients. Un alias mal fait peut empêcher le serveur apache de redémarrer, si c'est le cas vous trouverez les fichiers de configuration des alias dans `c:\wamp\alias`.*
6. Tester depuis votre client Web à l'URL : `http://localhost/inf340td/exercices/exercice1/test1.php`.

PHP est un préprocesseur appelé par le serveur Web lorsqu'il rencontre une page en `.php`. Dans la page en `.php` le préprocesseur traite tout ce qui est dans la balise `<?php ?>`. `phpinfo()` est une fonction qui donne des informations sur la configuration courante de PHP.

---

<sup>2</sup>Uniquement depuis PHP 5.3 et en CGI et FastCGI.

A titre indicatif, vous pouvez à l'aide de votre navigateur observer le fichier `http://localhost/inf340td/exercices/exercice1/test1.html` ainsi que son code source.

Avant d'aller plus loin, il vous faudra à la fin de chaque séance penser à exporter les fichiers de votre site sur votre partage NetBios.

## Important

Avant la fin de chaque séance de TP ou de TD, il vous faudra exporter votre projet :

1. Se placer sur le projet
2. Exporter
3. Général
4. Archive file

Pour l'import il vous faudra importer un projet existant dans le workspace courant.

## 2.2. PHP

Nos serveurs sont des serveurs de développement, nous pouvons choisir la version de PHP et modifier le `php.ini` (le fichier de configuration). Nous allons modifier la directive `error_reporting` qui a la valeur `E_ALL` et nous allons lui donner la valeur `E_ALL | E_STRICT` qui est la valeur conseillée pour le développement.

PHP est un langage qui a subi de nombreuses modifications, cette année nous allons utiliser la version 5.3 qui offre un meilleur support objet. En utilisant l'assistant graphique de WAMP, sélectionner la version 5.4 la plus récente<sup>3</sup>.

Une fois, la version 5.4 choisie, modifier en utilisant l'assistant graphique de WAMP votre `php.ini` la valeur de la directive `error_reporting` qui doit-être `E_ALL | E_STRICT`.<sup>4</sup>

Dans un développement normal; trois ensembles de serveurs sont utilisés avec chacun une configuration particulière :

les serveurs de développement

les serveurs de développement peuvent être personnel, ce qui est notre cas ou mutualisés. Ils sont dédiés au développement, souvent l'utilisateur dispose de droits étendus sur ces serveurs.

les serveurs de test

ce sont les serveurs qui permettent d'anticiper les problèmes de déploiement, ils permettent une validation par l'équipe et par les clients. Ils peuvent

être les dépôts d'un logiciel de gestion de versions.

les serveurs de productions

Ce sont les serveurs finaux, aucune modification ne devrait normalement y être faite.

Nous nous contenterons des serveurs de développement, mais pour vos projets tutorés, il ne vous faudra pas oublier la partie production sur laquelle il est déconseillé de faire des modifications. Les modifications en production ne sont pas compatibles avec un déploiement.

## 2.3. MySQL

Pour administrer MySQL nous allons utiliser une application Web : PHPMysqlAdmin.

1. Accéder à *PHPMysqlAdmin* avec l'URL : `http://localhost/phpmyadmin/`.

---

<sup>3</sup>Il n'est possible qu'il n'y ai qu'une version d'installée.

<sup>4</sup>Dans le `php.ini` ; marquer une ligne de commentaire

2. Exécuter le script `inf340tddb.sql` qui va créer la base `inf340tddb`, avec une table unique `utilisateur` qui contient deux colonnes `identifiant` et `mot_de_passe`.

Lors de la suite du développement vous pourrez utiliser PHPMyAdmin pour vérifier le résultat de vos scripts. Voici une explication sur le script MySQL que vous venez d'exécuter.

```
CREATE SCHEMA IF NOT EXISTS `inf340tddb`  
  DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci ;  
USE `inf340tddb` ;
```

`CREATE SCHEMA` permet de créer la base `inf340tddb` avec l'encodage `utf8`. `USE` permet de choisir la base de données par défaut.

```
CREATE TABLE IF NOT EXISTS `utilisateur` (  
  `identifiant` VARCHAR(45) NOT NULL ,  
  `mot_de_passe` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`identifiant`) )  
ENGINE = InnoDB;
```

`CREATE TABLE` permet de créer la table `utilisateur` avec ses deux colonnes `identifiant` et `mot_de_passe`, `identifiant` est une clef primaire.

```
INSERT INTO utilisateur (identifiant, mot_de_passe) VALUES ('admin','pass');  
INSERT INTO utilisateur (identifiant, mot_de_passe) VALUES ('root','password');
```

`INSERT INTO` permet d'insérer un tuple dans la table `utilisateur`.

```
CREATE USER 'etudiant'@'localhost' IDENTIFIED BY 'perdu';
```

`CREATE USER` permet de créer l'utilisateur `etudiant` avec comme mot de passe `perdu`.

```
GRANT ALL PRIVILEGES ON inf340tddb.* TO 'etudiant'@'localhost'  
WITH GRANT OPTION
```

`GRANT` permet d'allouer tous les privilèges pour un accès local au compte `etudiant` sur la base `inf340tddb`.

---

# Chapitre 3. Les TD, mon premier site Web

## 1. Présentation rapide du langage

Nous n'allons pas faire un cours détaillé de PHP mais simplement introduire les éléments qui nous seront utiles. Pour le moment nous savons qu'un script est compris dans la balise

```
<?php ?>
```

, il nous reste à trouver quoi y mettre. Les fonctions et les objets seront introduits plus finement ultérieurement.

### 1.1. Les types et variables

#### 1.1.1. Présentation

PHP est un langage faiblement typé, les variables ont un type défini par leur contenu, il n'est pas nécessaire de spécifier un type. Le type est défini par le contexte d'utilisation mais des transtypages sont possibles. PHP dispose de huit types :

4 scalaires

boolean

```
True, False
```

integer

entier relatifs, avec 0 comme préfix pour l'octal et 0x pour l'hexadécimal

float

0.1, 1.1e2, 4E-10

string

'une chaîne', "une chaîne avec évaluation des variables" ou encore avec la notation heredoc ou nowdoc

```
public $bar = <<<EOT  
bar  
EOT;
```

Voici un petit exemple :

### Exemple 3.1. Variables

```
<?php
$a;

var_dump($a); //NULL
$a=0;
var_dump($a); //int(0)
$a=$a+5;
var_dump($a); //int(5);
$a = $a + "un 10 zero";
var_dump($a); //int(5);

$a = $a+1.2;
var_dump($a); //float(6.2)

$a="toto";
var_dump($a); //string(4) "toto"
$a ='toto$a';
var_dump($a); //string(6) "toto$a"
$a = "toto$a";
var_dump($a); //string(7) "toto6.2"
//. est la concatenation
$a=$a."concat";
var_dump($a); //string(13) "toto6.2concat"

$b;
//isset permet de savoir si une variable est définie
var_dump(isset($b)); //bool(false)
$b = 1;
var_dump(isset($b)); //bool(true)

$c=true;
var_dump($c); //bool(true)
$c=True;
var_dump($c); //bool(true)
?>
```

Les affichages en php sont réalisés par `echo`, `print_r` ou `var_dump` :

`echo`

affiche un entier, un réel ou une chaîne de caractères,

`print_r`

affiche ce qu'affiche `echo` mais aussi les tableaux et les objets,

`var_dump`

offre un affichage pour le débogage.

#### 2 composés

Les types composés comportent :

les tableaux

qui sont des tableaux associatifs de taille variable introduits par le "constructeur" `array` :

```
array( clé => valeur , ... )
```

Voici un exemple d'utilisation de tableau :

### Exemple 3.2. Tableaux

```
<?php
$t = array(); //cree un tableau vide
$t[1]='un'; //associe à la clef 1 la valeur 1
$t['four']='quatre'; //associe à la clef 4 la valeur 4
$t['five']=5;
$t['clef']='valeur';
var_dump($t);
/*array(4)
 * { [1]=> string(2) "un"
 * ["four"]=> string(6) "quatre"
 * ["five"]=> int(5)
 * ["clef"]=> string(6) "valeur" }*/
echo count($t); //4

?>
```

#### les classes et les objets

les principes sont les mêmes que pour le langage java, mais PHP reste plus limité, notamment pour la surcharge et la redéfinition. Voici l'exemple de la classe Point et son utilisation :

### Exemple 3.3. Classe Point

```

class Point
{
    /**
     * @var unknown_type
     */
    private $x;
    private $y;

    public function __construct($x=0, $y=0){
        $this->x= (int) $x;
        $this->y= (int) $y;
    }

    public function __toString(){
        return "($this->x,$this->y)";
    }

    public function move($x,$y)
    {
        $this->x = (int) $x;
        $this->y= (int) $y;
    }
}

$p = new Point();

echo $p; //(0,0)
print_r ($p); //Point Object ( [x:private] => 0 [y:private] => 0 )
var_dump($p);
//object(Point)#1 (2) { ["x:private"]=> int(0) ["y:private"]=> int(0) }

$p->move(2,4);
print_r($p); //Point Object ( [x:private] => 2 [y:private] => 4 )
$p2=$p;
$p2->move(0,0);
print_r($p); //Point Object ( [x:private] => 0 [y:private] => 0 )

$p3=clone $p;
$p3->move(2,2);
print_r($p); //Point Object ( [x:private] => 0 [y:private] => 0 )
?>

```

Vous noterez au passage le constructeur (`__construct`), il existe de même les méthodes `__destruct()`, `__clone()`, ...

L'accès aux méthodes statiques (méthodes de classes) se fait en utilisant la syntaxe `NomClasse::NomMethodeStatique()`.

Lors d'un héritage, l'accès aux méthodes surchargées de la classe mère se fait en utilisant `parent::NomMethode`.

## 2 spéciaux

### Ressource

Une ressource est une variable spéciale, contenant une référence vers une ressource externe.

### NULL

En PHP une variable qui n'a pas encore reçue de valeur ou qui a été détruite(unset) est NULL.

La portée d'une variable dépend du contexte, dans un script c'est le script, dans une fonction c'est la fonction à moins que la variable ne soit définie comme étant globale. Les constantes peuvent aussi être statiques.

## 1.2. Les constantes

Les constantes en PHP sont introduites avec la fonction `define(nom, valeur)` ou avec le mot clef **const**.

### Exemple 3.4. Définition et utilisation d'une constante

```
const CONSTANT = 'Hello World';
define('CONSTANTE', "Bonjour le monde");

echo CONSTANT; //noter la disparition du $
echo CONSTANTE;
```

Il existe aussi des variables magiques qui renseignent sur l'état de l'interpréteur : `__FILE__` (le fichier courant), `__LINE__` (la ligne courante), ...

## 1.3. Les structures de contrôles

Sans faire de cours sur les expressions et sur les opérateurs de comparaison vous devez savoir que :

`$a == $b`  
est vraie si la valeur de `$a` est égale à la valeur de `$b`,

`$a === $b`  
est vraie si la valeur de `$a` est égale à la valeur de `$b` et si `$a` et `$b` sont de même type.

Vous pouvez utiliser les mêmes opérateurs logiques qu'en java : `&&` (AND), `||` (OR), `!` (Not).

### 1.3.1. Présentation

A l'exception des inclusions et du **foreach**, vous connaissez déjà les structures qui vont suivre, nous allons en donner un rappel sous forme de syntaxe et d'exemple.

#### 1.3.1.1. if, if else, if elseif

En PHP vous retrouverez à peu près les mêmes structures qu'en *java*, il existe cependant des écritures alternatives comme par exemple pour la conditionnelle :

```
(cond) ? commande1 : commande2 )
```

Nous reviendrons sur des structures alternatives .

#### if (expression) commandes

### Exemple 3.5. Exemple d'utilisation du if

```
if($a > $b)
    echo $a." est plus grand que ".$b;
```

#### if (expression) commandes else commandes

### Exemple 3.6. Exemple d'utilisation du if else

```
if($a > $b):
    echo $a." est plus grand que ".$b;
else
    echo $a. "est plus petit ou égale à ".$b;
```

#### if (expression) commandes elseif (expression) commandes

### Exemple 3.7. Exemple d'utilisation du elseif

```
if ($a > $b) {
    echo "a est plus grand que b";
} elseif ($a == $b) {
    echo "a est égal à b";
} else {
    echo "a est plus petit que b";
}
```



### 1.3.1.2. switch

Le switch est une structure pratique qui comme son nom l'indique permet un aiguillage sans un grand nombre de if.

#### Exemple 3.8. Exemple d'utilisation du switch

```
<?php
switch ($i){
    case 0:
        echo "i égal 0";
        break;
    case 1:
        echo "i égal 1";
        break;
    case 2:
        echo "i égal 2";
        break;
    default:
        echo "i n'est ni égal à 2, ni à 1, ni à 0";
}
?>
```

### 1.3.1.3. while, do while

Nous utiliserons les syntaxes **while (expression) commandes** et **do commandes while (expression);**, d'autres syntaxes existent en PHP.

#### Exemple 3.9. Exemple de boucle while

```
$i = 1;
while ($i <= 10) {
    echo $i++; /* La valeur affiche est $i avant l'incréméntation
                (post-incréméntation) */
}

```

#### Exemple 3.10. Exemple de boucle do while

```
$i = 0;
do {
    echo $i;
} while ($i > 0);
```

### 1.3.1.4. for, foreach

**for (expr1; expr2; expr3) commandes**

#### Exemple 3.11. Exemple de boucle for

```
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
```

La boucle **for** n'est pas simplement utilisable pour parcourir un tableau associatif, la boucle **foreach** répond à ce problème, la syntaxe est **foreach (array\_expression as \$value) commandes** ou **foreach (array\_expression as \$key => \$value) commandes**.

### Exemple 3.12. Exemple de boucle foreach

```

$sarr = array("un", "deux", "trois");
foreach ($sarr as $key => $value) {
    echo "Clé : $key; Valeur : $value<br />\n";
}
/*
Clé : 0; Valeur : un
Clé : 1; Valeur : deux
Clé : 2; Valeur : trois
*/

foreach ($sarr as $value) {
    echo "Valeur : $value<br />\n";
}
/*
Valeur : un
Valeur : deux
Valeur : trois
*/

$sarr = array('velo'=>'rouge', 'voiture'=>'bleu');
foreach ($sarr as $key => $value) {
    echo "Clé : $key; Valeur : $value<br />\n";
}
/*
Clé : velo; Valeur : rouge
Clé : voiture; Valeur : bleu
*/

foreach ($sarr as $value)
    echo "Valeur : $value<br />\n";
/*
Valeur : rouge
Valeur : bleu
*/

```

#### 1.3.1.5. require, include, require\_once, include\_once

Un script peut faire appel à d'autres scripts, **require** et **include** permettent de réaliser ces appels. **include** contrairement à **require** ne conduit pas à une erreur en cas de non chargement du script. D'inclusion en inclusion, une boucle peut se produire et des doubles définitions apparaître, **require\_once** et **include\_once**, ne conduisent pas à une erreur en cas de double définition.

#### 1.3.2. Les formes alternatives

PHP propose une autre manière de rassembler des instructions à l'intérieur d'un bloc, pour les fonctions de contrôle `if`, `while`, `for`, `foreach` et `switch`. Dans chaque cas, le principe est de remplacer l'accolade d'ouverture par deux points (`:`) et l'accolade de fermeture par, respectivement, `endif;`, `endwhile;`, `endfor;`, `endforeach;`, ou `endswitch;`(php.net).

Cette approche est intéressante dans les vues, comme par exemple lors de l'intégration de design dans un CMS.

```
<?php if ($a == 5): ?> A égal 5 <?php endif; ?> est équivalent à <?php if ($a == 5) echo "A égal 5"; ?>
```

#### 1.3.3. Exercices

Vous allez travailler dans le répertoire `exercices\exercice2`.

##### 1.3.3.1. Création d'un tableau

Dans le fichier `exercice2.1` créer un tableau nommé `$utilisateurs` et dont le `var_dump` doit donner le résultat suivant :

```
array (size=2)
  0 =>
    array (size=2)
```

```
'identifiant' => string 'admin' (length=5)
'mot_de_passe' => string 'pass' (length=4)
1 =>
array (size=2)
'identifiant' => string 'root' (length=4)
'mot_de_passe' => string 'password' (length=8)
```

Ce tableau est un tableau *numéroté* de deux lignes dont la première et la seconde ligne contiennent un tableau *associatif*, bref un tableau de tableau.

### 1.3.3.2. Inclure un fichier

Supprimer dans `exercice2.1` le `var_dump`, la consultation de `http://localhost/inf340td/exercices/exercice2/exercice2.1.php` ne doit plus produire d'affichage. Dans `exercice2.2` inclure (**require** ou **include** ou **require\_once** ou **include\_once**) le fichier `exercice2.1` et faire afficher avec un `print_r` le tableau `$utilisateurs`. Vous devez obtenir :

```
Array (
[0] => Array ( [identifiant] => admin [mot_de_passe] => pass )
[1] => Array ( [identifiant] => root [mot_de_passe] => password )
)
```

### 1.3.3.3. Les boucles

Dans `exercice2.3` inclure le fichier `exercice2.1` et en utilisant deux boucles imbriquées, une pour les lignes et une pour les colonnes, produire la page dont le code source XHTML est le suivant :

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<title>TD inf340</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<table border="1">
<thead>
<tr> <th> identifiant </th> <th> mot de passe </th> </tr>
</thead>
<tbody>
<tr>
<td> admin</td>
<td> pass</td>
</tr>
<tr>
<td> root</td>
<td> password</td>
</tr>
</tbody>
</table>
</body>
</html>
```

### 1.3.3.4. Les conditionnelles

Dans `exercice2.4` vous devez parcourir le tableau `utilisateurs` et afficher trop court si la longueur du mot de passe est inférieure à 8 et Ok sinon.

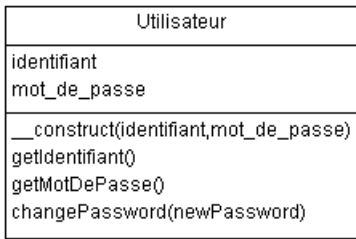
```
pass trop court password OK
```

La fonction `strlen()` permet de connaître la longueur d'une chaîne.

### 1.3.3.5. Les classes et les objets

Dans `exercice2.5` vous allez coder la classe `Utilisateur` correspondant au schéma UML suivant :

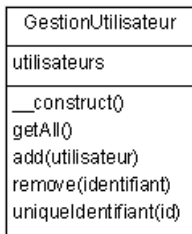
**Figure 3.1. Diagramme de classe de la classe Utilisateur**



Un programme de test vous est fourni dans `exercice2.5b`.

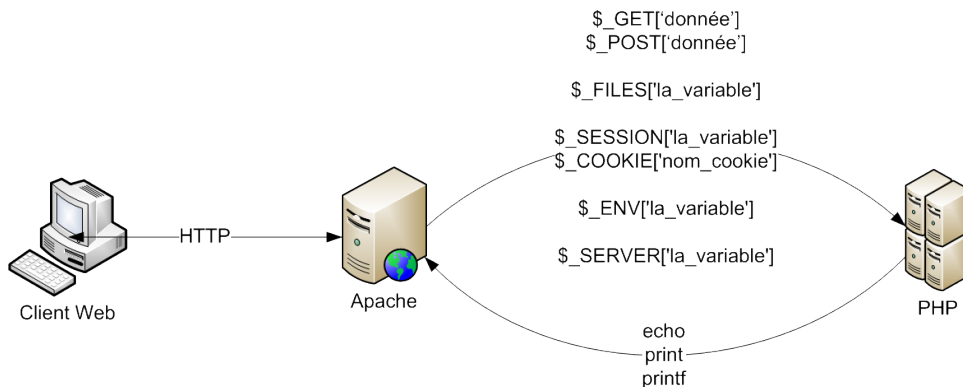
Le fichier `exercice2.6` contient la classe `GestionUtilisateur` à vous de réaliser dans `exercice2.6b` le programme de test. Ce programme devra tester chacune des méthodes publiques.

**Figure 3.2. Diagramme de classe de la classe GestionUtilisateur**



## 2. La communication entre le serveur Web et PHP

**Figure 3.3. Communication entre le serveur Web et PHP**



Le serveur Web et PHP communique, PHP reçoit des informations du serveur Web qui lui même en tient une partie du client. Pour ce qui nous intéresse ces informations sont dans les tableaux associatifs : `$_GET`, `$_POST` et `$_REQUEST` (un tableau associatif qui contient par défaut le contenu des variables `$_GET`, `$_POST` et `$_COOKIE`). L'écriture est réalisée avec **echo** ou **print** ou **printf**.

### 2.1. Exercices

L'information provient principalement de formulaires ou d'ancres. Nous allons mettre en oeuvre ces deux types d'envois d'information.

#### 2.1.1. Communication en POST

Dans ce premier exercice, vous disposez d'un formulaire `exercice3.1.html` et vous devez faire afficher les valeurs saisies. Le code du formulaire est le suivant :

```
<form action="exercice3.1.php" method="post">
  <p> identifiant : <input type="text" name="identifiant"/> </p>
  <p> password : <input type="password" name="mot_de_passe"/> </p>
  <p> <input type="submit"/>
</form>
```

Pour rappel, *action* représente l'URL où sera envoyée l'information, l'attribut *name* définit le nom de l'information.

Dans `exercice3.1.php` faire afficher l'identifiant et le mot de passe saisis.

### 2.1.2. Communication en GET

L'exercice est en deux parties, la première repose sur l'utilisation d'`exercice3.2.php`, ce script affiche sous la forme d'un tableau le contenu du tableau `$utilisateurs`. Les ancres ont un href vide à vous de faire en sorte qu'elles aient la forme :

- `http://localhost/inf340td/exercices/exercice3/exercice3.2b.php?action=update&id=identifiant_choisi,`
- `http://localhost/inf340td/exercices/exercice3/exercice3.2b.php?action=update&id=identifiant_choisi.`

L'identifiant choisi est celui de l'utilisateur présent sur la même ligne du tableau, en effet, avant de supprimer ou de modifier un utilisateur, il est intéressant de le connaître et sur notre exemple, il est connu par son identifiant. Votre code xhtml doit ressembler à ceci :

```
...
  <tr>
    <td> admin</td>

    <td> pass</td>

    <td> <a href="exercice3.2b.php?action=update&id=".pass> U </a> </td>
    <td> <a href="exercice3.2b.php?action=delete&id=".pass> D </a></td>
  </tr>

  <tr>
    <td> root</td>

    <td> password</td>

    <td> <a href="exercice3.2b.php?action=update&id=".password> U </a> </td>
    <td> <a href="exercice3.2b.php?action=delete&id=".password> D </a></td>
  </tr>
...

```

Dans le fichier `exercice3.2b.php` faites dans un premier temps afficher l'identifiant et l'action puis dans un second temps faites afficher "mode suppression" si l'action est `delete` et "mode modification" si l'action est `update`. Vous devez avoir par exemple :

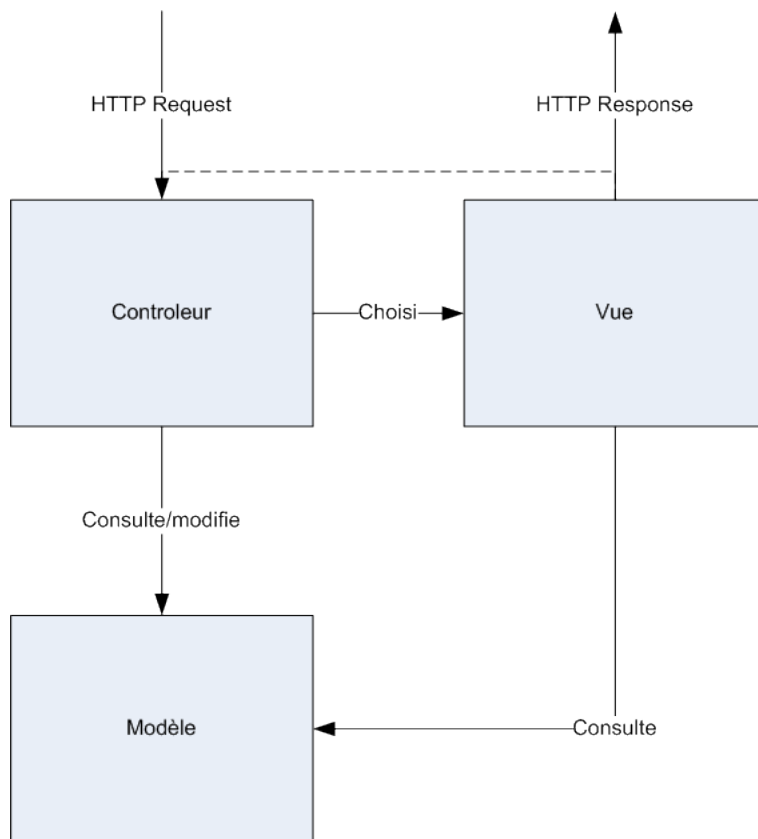
```
mode suppression pour admin
```

Exception faite de l'accès aux bases de données, nous avons le minimum qui va nous permettre de réaliser un site. Un premier site sera réalisé dans la section suivante.

## 3. Le MVC (Modèle Vue Contrôleur)

Le patron de conception MVC permet de séparer les métiers du Web et un travail en équipe, l'informaticien peut se concentrer sur les modèles et sur les contrôleur, le Web Designer et l'intégrateur Web sur les vues.

**Figure 3.4. MVC en PHP**



Séparer les concepts permet de séparer les métiers et faciliter la réutilisation. Dans ce patron de conception nous avons :

la vue

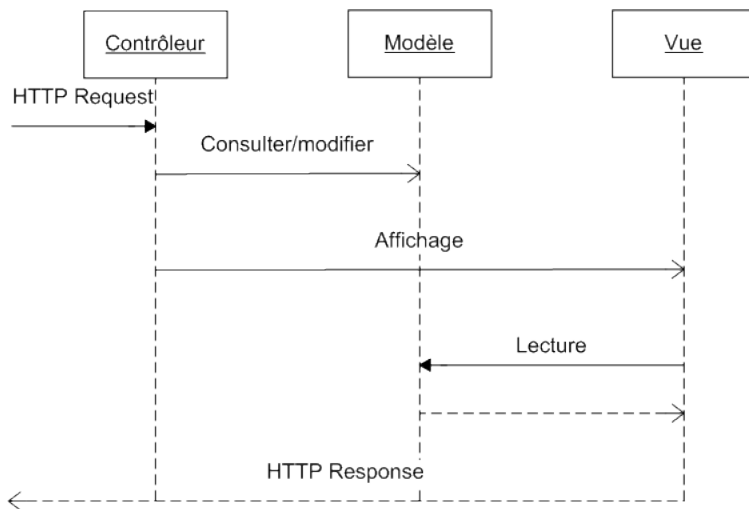
qui est l'interface utilisateur, elle présente le modèle, elle reçoit les événements pour en avertir le contrôleur. Elle ne modifie pas le modèle et elle n'effectue aucun traitement.

le modèle

qui définit le comportement de l'application et le traitement des données. Les données ne sont associées à aucune présentation. Le modèle assure la cohérence des données et offre des méthodes de manipulation de ces dernières. Dans une architecture non Web, le modèle peut changer ses changements à la vue.

le contrôleur

qui gère les événements en modifiant le modèle et en invoquant les vues.

**Figure 3.5. MVC diagramme de séquence**

Dans les architectures Web PHP, nous retrouvons deux types de contrôleurs :

le contrôleur frontal ou principale

il reçoit les requêtes et sollicite les contrôleurs de page, c'est le seul point d'entrée du système, tout passe par lui. C'est un singleton généralement nommé `index.php`. Le choix du contrôleur de page, de l'action et des paramètres est réalisé grâce à l'URL.

le contrôleur de page

il gère sous commande du contrôleur frontal une partie du modèle

## 4. Le développement

Nous avons pour le moment notre serveur apache configuré ainsi que notre SGBD. Nous allons maintenant nous concentrer sur le PHP et commencer par le contrôleur principal. Nous utiliserons une approche de conception incrémentale ou à chaque étape, nous ajouterons de nouvelles fonctionnalités. Cette approche est radicalement différente du cycle en V où l'intégration est réalisée à la fin. Vous aurez besoin des documentations disponibles à l'URL suivante : <http://www.php.net/manual/fr/index.php>.

### 4.1. Le contrôleur principal et les constantes

Nous allons commencer par le fichier `index.php`, ce fichier placé à la racine devra être utilisé dans toutes les URL, c'est le script PHP qui peut-être utilisé directement dans une URL.

Le contrôleur principal est très simple, il inclut des constantes et passe la main au contrôleur de page : `utilisateur.php`<sup>1</sup>.

Le code de notre contrôleur principal est le suivant :

```

<?php
define('FCPATH', realpath(dirname(__FILE__)).DIRECTORY_SEPARATOR);
define('APPPATH', FCPATH.'application'.DIRECTORY_SEPARATOR);

require_once APPPATH.'config'.DIRECTORY_SEPARATOR.'constants.php';
require_once CONTROLLERSPATH.'utilisateur.php';

?>
  
```

La première constante `FCPATH` est définie automatiquement, elle est obtenue à partir du chemin physique de `index.php` (le script en cours d'exécution). Le point (`.`) réalise la concaténation et la constante prédéfi-

<sup>1</sup>Ici, nous n'avons qu'un seul contrôleur de page.

Le `DIRECTORY_SEPARATOR` représente le séparateur de répertoire (`\` sous Windows, `/` sous Linux). Si `FC-PATH` vaut `D:\workspaces\workspacePHP\inf340_td\` alors `APP_PATH` devra valoir `D:\workspaces\workspacePHP\inf340_td\application\`.

La structure de notre projet est :

```
| -site
| -index.php
| -application
|   | -config
|   | -controllers
|   | -models
|   | -views
| -ressources
```

Pour faciliter l'accès au répertoire `application` une constante `APP_PATH` est définie.

L'inclusion du fichier `constants.php` permet de rendre visible les constantes qui y seront définies dans toutes les pages du site. Et enfin, le contrôleur de page `utilisateur.php` est chargé. *Il vous faut préférer `require_once` ou `include` aux `include`, ainsi un fichier non trouvé provoque l'arrêt du script.*

Notre fichier de constante contient :

- des constantes pour faciliter l'accès aux répertoires : `CONTROLLERS_PATH`, `VIEWSPATH`, `MODELS_PATH`, `TEMPLATES_PATH`
- des constantes pour faciliter l'accès aux bases de données : `DB_HOST` (la machine serveur SGBD), `DB_NAME` le nom de la base, `DB_LOGIN` le login d'accès à la base, `DB_PASSWORD` le mot passe associé.

Normalement, si vous avez suivi la procédure, le fichier est déjà préconfiguré, seul l'URL est peut-être différente.

## 4.2. Le contrôleur de page utilisateur.

Le contrôleur de page utilisateur (`utilisateur.php`) est responsable des actions sur un utilisateur ou des utilisateurs, nous avons :

`read`

qui permet d'afficher la page d'accueil,

`create`

qui permet d'ajouter un utilisateur,

`delete`

qui permet de supprimer un utilisateur,

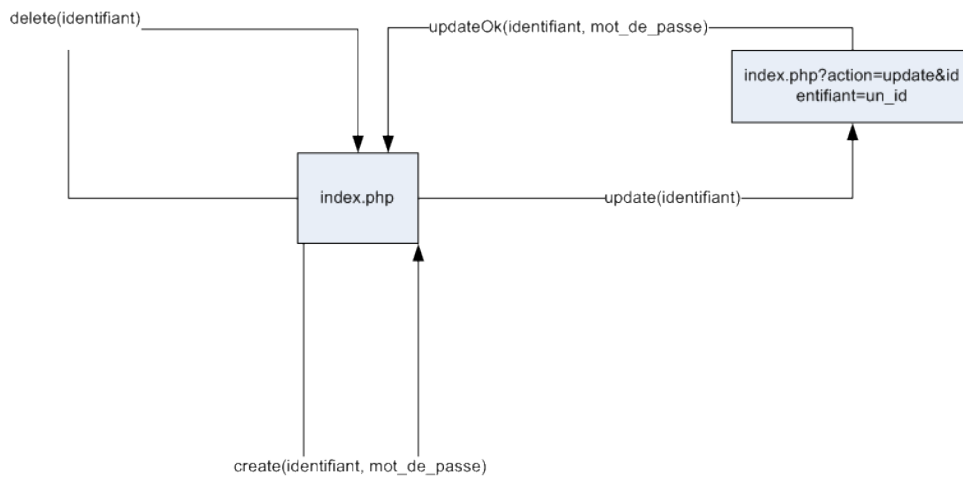
`update`

qui permet d'afficher un formulaire de modification pour un utilisateur choisi.

`updateOk`

qui permet de modifier l'utilisateur choisi.



**Figure 3.6. Navigation**

La convention choisie pour nos URL est `index.php?action=xxx&identifiant=yyy`.

Une ossature de code vous est fournie :

```

<?php
if (isset($_GET['action']))
$action = $_GET['action'];
else
if (isset($_POST['action']))
$action = $_POST['action'];
else
$action='read';

switch ($action) {

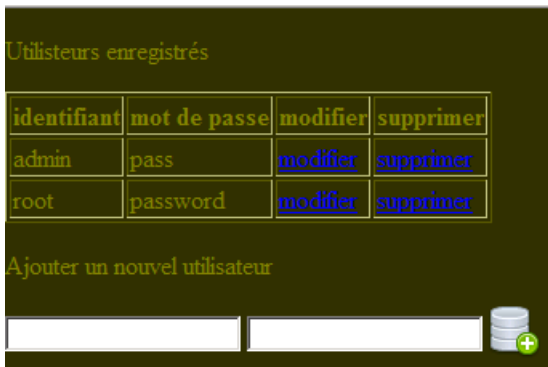
case ('create'):
//ici si action=create
break;
case 'read':
//ici si action=read ou si action n'est pas positionné
break;
case 'update':
//ici si action=update
break;
case 'updateOk':
//ici si action=updateOk
break;
case 'delete':
//ici si action=delete
break;
default:
;
break;
}
?>

```

En fonction de l'action choisie un 'case' est utilisé.

### 4.3. La page d'accueil pour afficher l'ensemble des utilisateurs

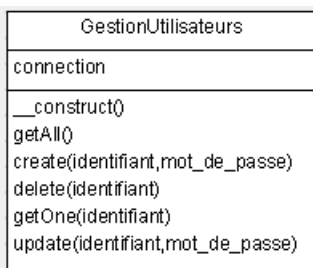
La page d'accueil correspond à l'action `read`, le contrôleur sollicite le modèle pour obtenir les utilisateurs puis affiche une vue.

**Figure 3.7. La page d'accueil**

### 4.3.1. Une courte introduction à PDO (PHP Data Objects)

PDO est une interface d'accès aux bases de données fournie en standard avec PHP. Nous allons l'utiliser pour nos requêtes. Vous trouverez la documentation à l'URL suivante : <http://php.net/manual/fr/intro.pdo.php> et un cours à l'URL : <http://studio.jacksay.com/tutoriaux/php/connection-mysql-avec-pdo>.

Vous disposez déjà la classe `GestionUtilisateurs` présente dans le répertoire `models`.

**Figure 3.8. Le diagramme de classe de la classe `GestionUtilisateurs`**

Pour accéder à une base de données, il faut se connecter en passant la machine, la base, le login et le mot de passe et éventuellement d'autres options comme ici un affichage détaillé des erreurs :

```
$dns = 'mysql:host='.DB_HOST.';dbname='.DB_NAME;
$this->connection = new PDO( $dns, DB_LOGIN, DB_PASSWORD );
$this->connection->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
```

La connexion pouvant ne pas réussir, il faut gérer l'erreur, c'est ici le rôle du bloc `try/catch`. Si une erreur se produit dans le bloc `try` alors l'exécution se poursuit dans le bloc `catch` correspondant.

```
try {
//code susceptible de lever une erreur
}
catch(ClasseException1 $e){//traitement de l'exception}
catch(ClasseException2 $e){//traitement de l'exception}
}
```

Une fois la connexion obtenue, nous allons pouvoir réaliser des requêtes, c'est ce que nous illustrerons dans les exemples qui vont suivre.

### 4.3.2. Le modèle

Dans notre page d'accueil, nous souhaitons faire afficher l'ensemble des utilisateurs, il nous faut donc interroger la base de données pour obtenir la liste des utilisateurs. La méthode `getAll()` est dédiée à cet effet :

```
$req = $this->connection->query("SELECT identifiant, mot_de_passe FROM utilisateur");
```

```
$utilisateurs = $req->fetchAll(PDO::FETCH_OBJ);
```

Depuis, notre connexion, nous exécutons une requête (méthode `query()`) puis nous récupérons l'ensemble des enregistrements avec `fetchAll()` (`fetch()` permet de ne récupérer qu'un seul enregistrement). Le paramètre `PDO::FETCH_OBJ` permet d'obtenir un résultat sous forme d'un tableau d'objets.

Utiliser le fichier `tests/test_bd1.php` pour tester la méthode `getAll()`. Avec un `var_dump()`, vous devez obtenir :

```
array (size=2)
  0 =>
    object(stdClass) [4]
      public 'identifiant' => string 'admin' (length=5)
      public 'mot_de_passe' => string 'pass' (length=4)
  1 =>
    object(stdClass) [5]
      public 'identifiant' => string 'root' (length=4)
      public 'mot_de_passe' => string 'password' (length=8)
```

Disposant des données, nous allons chercher à les afficher dans une vue, mais la vue ne pouvant voir directement le modèle, nous allons passer par un contrôleur.

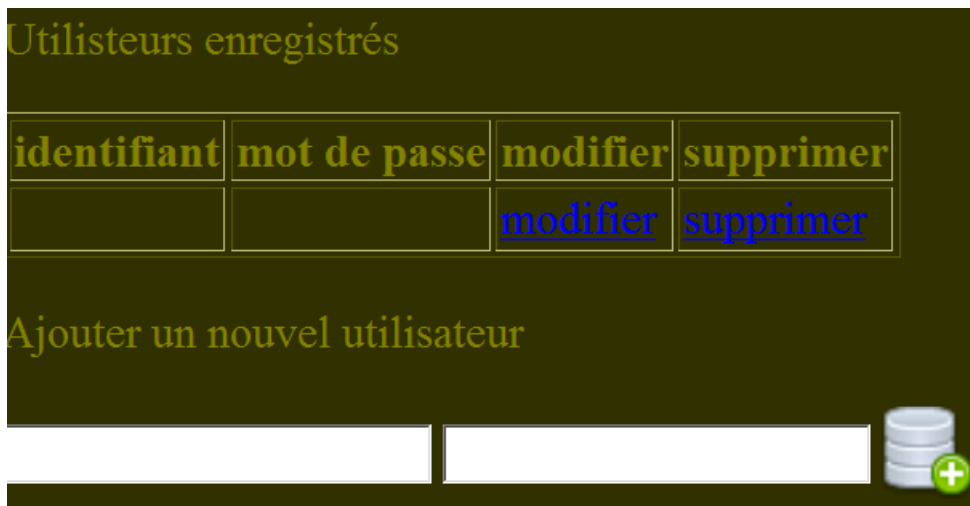
### 4.3.3. Le contrôleur

Pour la méthode `read`, le contrôleur va solliciter le modèle pour récupérer les utilisateurs, les stocker dans une variable, `$utilisateurs` par exemple et les passer à la vue (via un `require` ou un `include`), ici `accueil_view.php`.

A vous de réaliser les opérations précédemment décrites<sup>2</sup>.

Vous devez pour les URL suivantes : `http://localhost/inf340td/site/`, `http://localhost/inf340td/site/index.php`, `http://localhost/inf340td/site/index.php?action=read` obtenir :

**Figure 3.9. La page d'accueil sans ses données**



### 4.3.4. La vue

La vue a pour objectif d'interagir avec l'utilisateur, pour le moment, elle nous permet d'afficher sous forme d'un tableau les utilisateurs récupérés par le contrôleur au près du modèle et fournies dans la variable `$utilisateurs`.

Faites en sorte que les utilisateurs puissent être affichés par la vue `accueil_view.php`.

La syntaxe alternative de la boucle `foreach` peut vous être utile : `http://www.php.net/manual/fr/control-structures.foreach.php`.

<sup>2</sup>Souvent les pages partagent la même css et le même JS, c'est pourquoi on définit des templates.

Nous nous occuperons ultérieurement des liens pour le moment votre html doit ressembler à ceci :

```
<tbody>
  <tr>
    <td> admin</td>
    <td> pass</td>
    <td> <a href=""> modifier </a></td>
    <td> <a href=""> supprimer </a></td>
  </tr>
  <tr>
    <td> root</td>
    <td> password</td>
    <td> <a href=""> modifier </a></td>
    <td> <a href=""> supprimer </a></td>
  </tr>
</tbody>
```

## 4.4. L'ajout d'un utilisateur

Pour ajouter un formulaire, nous allons utiliser le formulaire en bas de la vue `accueil_view.php`, son action est `http://localhost/inf340td/site/index.php?create`, sa méthode est *post* et l'information envoyée est *identifiant* et *mot\_de\_passe*.

Nos requêtes peuvent avoir besoin de paramètres, comme ici l'identifiant et le mot de passe, il nous faut donc introduire la notion de requêtes paramétrées.

### 4.4.1. Une courte introduction aux requêtes paramétrées

Comme vous le savez déjà, la requête utilisée pour ajouter des données est `INSERT INTO`, nous allons paramétrer cette requête :

```
$req = $this->connection->prepare(
    "INSERT INTO utilisateur (identifiant, mot_de_passe) VALUES (?,?)"
);
```

Vous observerez la présence de deux `?`, se sont les paramètres, nous les fournirons à l'exécution :

```
$res= $req->execute(array($identifiant, $mot_de_passe));
```

Vous observerez aussi dans la méthode `create()` la présence d'un bloc `try/catch`, en effet si l'on essaye d'insérer deux fois le même identifiant une contrainte d'unicité est levée, `identifiant` étant une clef primaire.

### 4.4.2. Le modèle

Vous disposez du formulaire `tests/test_bd2.html`, modifier le script `tests/test_bd2.php` pour que via la méthode `create()` il insère les valeurs fournies par le formulaire.

### 4.4.3. Le contrôleur

Tout comme dans le test précédent, le contrôleur doit récupérer en `post` (`$_POST`) l'identifiant (`$_POST['identifiant']`) et le mot de passe (`$_POST['mot_de_passe']`), solliciter le modèle et réafficher la page d'accueil comme suit.

### 4.4.4. La vue

Pour réafficher la page d'accueil vous aller informer le client web que `http://localhost/inf340td/site/inde.php?create` n'est pas disponible mais que `http://localhost/inf340td/site/inde.php?read` l'est. Cette opération est réalisée en utilisant l'entête HTTP, soit en PHP : `header('Location: '.URL.'index.php?action=read')`.

## 4.5. La suppression d'un utilisateur

Pour supprimer un utilisateur, il faut le connaître, un moyen simple est de modifier la vue `accueil_view.php` pour que chacune des ancres contienne comme href `http://localhost/inf340td/site/index.php?action=delete&identifiant=valeur` où valeur est un identifiant. Faites ce premier travail et vérifiez l'HTML obtenu. *Préférer les constantes au nom du site.*

### 4.5.1. Le modèle

Compléter la méthode `delete($identifiant)` du modèle pour supprimer un nouvel utilisateur.

### 4.5.2. Le contrôleur

L'information vient d'une ancre, il faut donc récupérer l'identifiant en utilisant `$_GET` puis solliciter le modèle et réafficher la vue `accueil_view.php`.

### 4.5.3. La vue

La vue est réaffichée comme précédemment en utilisant une redirection (`header location`).

## 4.6. La modification d'un utilisateur

La modification d'un utilisateur est plus compliquée, elle repose sur l'utilisation de deux vues : `accueil_view.php` et `update_view.php`. Il faut que l'identifiant de l'utilisateur à modifier choisi dans `accueil_view.php` soit passé via le contrôleur à la vue `update_view.php` (le formulaire de modification), la méthode `update` du contrôleur `utilisateur.php` est là pour cela. La vue `update_view.php`, elle sollicitera la méthode `updateOk` du contrôleur.

### 4.6.1. La vue `accueil_view.php`

Tout comme pour la suppression, il vous faut modifier les URL : `http://localhost/inf340td/site/index.php?action=update&identifiant=valeur`.

### 4.6.2. Le modèle

Pour la modification, il nous faut deux méthodes :

`getOne($identifiant)`

qui à partir de l'identifiant retourne l'utilisateur correspondant. Penser à utiliser `fetch()` plutôt que `fetchAll()`.

`update($identifiant, $mot_de_passe)`

qui modifie le mot de passe de l'utilisateur correspondant à l'identifiant.

Si vous souhaitez réaliser des scripts de test avant de passer au contrôleur, ce n'est pas interdit.

### 4.6.3. La méthode `update` du contrôleur

Cette méthode reçoit en `GET` (d'une ancre) l'identifiant puis sollicite le modèle (`getOne($identifiant)`) pour obtenir l'utilisateur correspondant puis enfin affiche la vue `update_view.php`.

### 4.6.4. La vue `update_view.php`

C'est un formulaire que vous avez à réaliser et qui affiche l'utilisateur précédemment sélectionné. L'action du formulaire est `http://localhost/inf340td/site/index.php?action=updateOk`. Le champ identifiant est en `readonly`.

### 4.6.5. La méthode `updateOk` du contrôleur

Elle reçoit en `POST` l'identifiant et le mot de passe venant de la vue `update_view.php` sollicite le modèle (`update($identifiant, $mot_de_passe)`) puis redirige vers la page d'accueil.

---

# Chapitre 4. Les TP, utilisation d'un framework et d'un ORM

Avant de commencer cette partie vous devez avoir défini vos groupes pour le devoir maison ainsi que le sujet.

Nous allons utiliser l'ORM (Object-Relational Mapping) Doctrine et le framework *CodeIgniter*, pour réaliser un site de galeries photo.

- tous les utilisateurs peuvent avoir une galerie,
- les utilisateurs de niveau 0 (administrateurs) peuvent créer et supprimer des utilisateurs.

CodeIgniter est basé sur le design pattern MVC et sur le design pattern *ActiveRecord*, cependant nous n'utiliserons pas les modèles fournis avec CodeIgniter mais nous utiliserons L'ORM Doctrine et les annotations.

Les ORM ont pour objectif, de créer une correspondance entre le monde des objets et le monde relationnel. En attribuant une classe à une table et un attribut de la classe à un champ de la table, il devient possible d'assurer l'accès au données et la persistance simplement en utilisant des objets. Cette approche qui existe depuis longtemps dans le monde Java (Hibernate, Java Persistence API) où dans le monde du .net (NHibernate, Language INtegrated Query) est maintenant disponible en PHP, vous allez voir que dans la majorité des cas, vos objets vous suffiront, vous n'aurez plus besoin du SQL.

Le mapping objet relationnel objet peut-être réalisé de trois façons en doctrine, avec :

des annotations

```
<?php

/**
 * @Entity * @Table(name="my_persistent_class")
 */

class MyPersistentClass { //... }
```

des fichiers XML

```
<doctrine-mapping>

<entity name="MyPersistentClass" table="my_persistent_class">

<!-- ... -->

</entity>

</doctrine-mapping>
```

des fichiers YAML

```
MyPersistentClass:

  type: entity

  table: my_persistent_class

# ...
```

Les trois exemples précédents indique que la classe `MyPersistentClass` est le mapping de la table `my_persistent_class`. Dans les TP, nous utiliserons les annotations, cette approche a pour avantage de ne pas créer de fichier supplémentaire.

Nous travaillerons avec la dernière version stable de Doctrine, la 2.0, vous aurez besoin de la documentation suivante : <http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/index.html> et plus par-

ticulièrement de la partie référence. Pour CodeIgniter, vous pourrez utiliser la documentation fournie avec le projet dans le répertoire `user_guide`.

## 1. Mise en place

Nous aurons comme en TD à mettre en place :

le PHP

importer dans eclipse le projet (`inf340_tp`), nous modifierons les fichiers de configuration après.

la base de donnée

En utilisant *phpMyAdmin* créer une base de données nommée *inf340tpdb*, en *utf8-general*. Vous devez posséder un compte MySQL ayant tous les droits sur cette base. Vous pouvez, si vous le souhaitez, utiliser le script fourni. Nous générerons les tables et leurs contenus depuis PHP.

la configuration du serveur Web

Elle est réalisée en deux temps, en créant un alias nommé *inf340tp* et portant vers le répertoire de votre projet, il contient le contrôleur principal : `index.php` qui sera l'entrée de notre site ; puis en modifiant le fichier `.htaccess` à la racine du projet. Ce dernier fichier permet si `index.php` n'apparaît pas dans l'URL de le faire rajouter par apache avant de le passer à PHP<sup>1</sup>. Il permet aussi avec l'option `MultiViews` de négocier le contenu. La sélection ou négociation de contenu, consiste en la sélection d'une version de document qui correspond le mieux aux possibilités du client, parmi un ensemble de documents. Nous l'utiliserons ainsi, si une ancre référence un nombre alors ce nombre `.jpg` ou `.png` sera servi. Dans nos URL, nous n'indiquerons pas les extensions des fichiers images.

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule ^(.*)$ /inf340tp/index.php/$1 [L]
</IfModule>
<IfModule !mod_rewrite.c>
  ErrorDocument 404 /inf340tp/index.php
</IfModule>
Options +MultiViews
```

Le fichier `config.php` doit aussi être modifié `$config['index_page'] = 'index.php'`; doit-être modifiée par `$config['index_page'] = ''`; Dans le projet fourni la modification est déjà faite.

La consultation de `http://localhost/inf340tp/index.php` ou de `http://localhost/inf340tp` doit conduire à la page d'accueil de CodeIgniter.

## 2. Développement

Nous commencerons par un peu de lecture sur CodeIgniter avant de créer à partir de notre modèle, nos tables et les peupler.

### 2.1. Les principaux fichiers de CodeIgniter

Vous retrouver comme en TD à la racine un fichier nommé *index.php* c'est le contrôleur principal, il va aiguiller vers les contrôleurs d'application présents dans le répertoire `application`. Le répertoire `system` contient les *libraries* et "*helpers*" de CodeIgniter, les *libraries* sont des classes et les *helpers* sont des collections de fonctions. Le répertoire `user_guide` contient la documentation, vous pouvez l'effacer une fois le développement fini. Le répertoire `application`, contient votre application et enfin le répertoires `resources` non fourni avec CodeIgniter contient les ressources du site (images, CSS, ...).

Dans le répertoire `application` vous trouverez un répertoire pour les contrôleurs (`controllers`), un pour les modèles (`models`) et enfin un pour les vues (`views`).

---

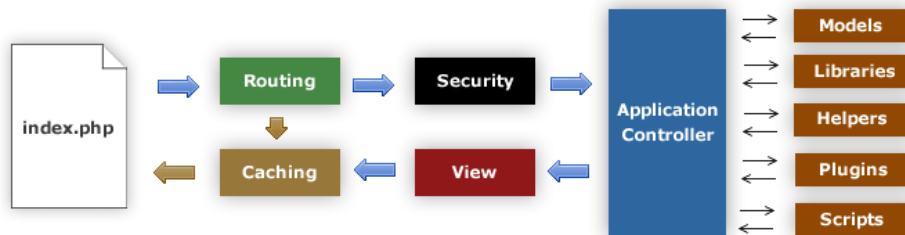
<sup>1</sup>Il faut que `mod_rewrite` d'apache soit activé.

Les fichiers de configuration se trouvent dans le répertoire `config` d'application, dans `config.php` vous aurez à placer l'URL du site et à modifier l'"*index File*" Il doit être vide ici, nous avons grâce au fichier `.htaccess` supprimé `index.php` de l'URL., dans `database.php` les informations sur la base de données, et enfin dans `routes.php` le contrôleur par défaut pour nous ce sera `accueil`.

A vous de modifier, si nécessaire `config.php`, `database.php` et `route.php`.

L'URL `http://localhost/inf340tp/user_guide/` vous conduit à la documentation.

**Figure 4.1. Les modules de CodeIgniter**



## 2.2. Notre modèle

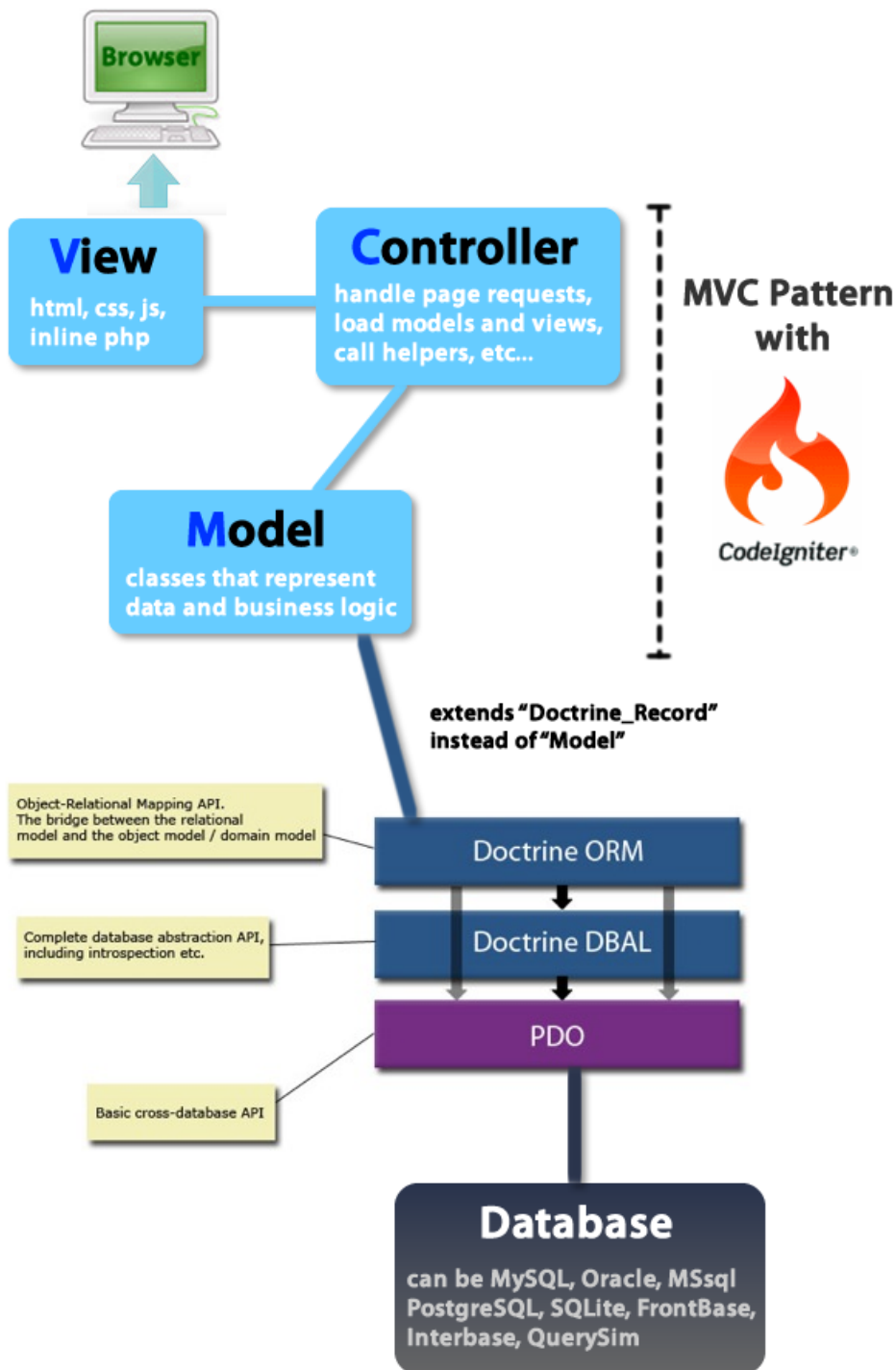
### Important

Attention, nous n'utiliserons pas des modèles reposant sur les classes de *CodeIgniter* mais sur ceux de *Doctrine*, ne lisez donc pas la documentation de *CodeIgniter* sur les modèles mais celle de *Doctrine*.

Dans le répertoire `models` vous trouverez les entités (les classes associées aux tables), elles utilisent les annotations, les composants métiers seront dans le répertoire `repositories` ils permettent la manipulation des entités et enfin vous trouverez un répertoire `proxies` qui contiendra des classes générées par *Doctrine* pour faciliter les requêtes multi-tables. Par exemple, comme les utilisateurs possèdent plusieurs images (`@OneToMany(targetEntity="Image", mappedBy="utilisateur")`), vous pourrez directement à partir d'un utilisateur accéder à ses images.



Figure 4.2. Code Igniter et Doctrine



### 2.2.1. Étude de notre modèle

Un première version des entités vous est fournie : `Image.php` et `Utilisateur.php`, en utilisant Doctrine essayer de comprendre les annotations utilisées.

Dans `Utilisateur.php` la librairie `encrypt` de CodeIgniter est utilisée, vous pouvez regarder la documentation de CodeIgniter pour en comprendre le fonctionnement.

## 2.2.2. Création de nos tables

Le contrôleur `install/install.php` permet de créer vos table en lisant la documentation de CodeIgniter sur les contrôleurs, à vous de trouver comment l'utiliser, vous pouvez observer le résultat avec *phpMyAdmin*. Vous devez obtenir le SQL suivant :

```
CREATE TABLE IF NOT EXISTS `image` (  
  `url` int(11) NOT NULL AUTO_INCREMENT,  
  `utilisateur` varchar(45) DEFAULT NULL,  
  `description` varchar(512) DEFAULT NULL,  
  PRIMARY KEY (`url`),  
  KEY `IDX_C53D045F1D1C63B3` (`utilisateur`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;  
  
CREATE TABLE IF NOT EXISTS `utilisateur` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `login` varchar(45) NOT NULL,  
  `level` int(11) DEFAULT NULL,  
  `password` varchar(45) NOT NULL,  
  `description` varchar(128) NOT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `UNIQ_1D1C63B3AA08CB10` (`login`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=3 ;  
  
ALTER TABLE `image`  
  ADD CONSTRAINT `FK_C53D045F1D1C63B3` FOREIGN KEY (`utilisateur`) REFERENCES `utilisateur` (`id`)  
  ON DELETE CASCADE;
```

## 2.2.3. Le peuplement de nos tables

Modifier `install.php` pour, sans faire appel aux dépôts mais en vous inspirant, rajouter dans la base des utilisateurs, au moins un par niveau `$em = $this->doctrine->em; permet dans un contrôleur de récupérer le gestionnaire d'entité, ce dernier vous permet de rendre vous entités persistantes $em->persist($utilisateur) et réaliser un commit $em->flush() plus tard, vous vous servirez pour charger les dépôts $em->getRepository('models\Utilisateur')`.

Vous pourrez observer le résultat avec *phpMyAdmin*.

## 2.3. Le frontoffice premier partie

Le frontoffice est ce que perçoit le client.

Le contrôleur concerné est `Accueil`, c'est un contrôleur simple, il étend `CI_Controller`, nous utiliserons plus tard un contrôleur sécurisé avec authentification.

Pour l'URL `http://localhost/inf340tp/accueil` : vous devez avoir l'affichage suivant :



Modifier `route.php` pour que l'URL `http://localhost/inf340tp/` conduise au même affichage.

### 2.3.1. Affichage de la liste des galeries (`accueil_view`)

Modifier `Accueil` pour que la vue `accueil_view` soit affichée.

#### Important

Vous observerez que la méthode `findAll` n'est pas redéfinie dans le dépôt (`UtilisateurRepository`), elle est héritée, ainsi vous disposez aussi de `findByXXX`.

Vous devez maintenant avoir :



### 2.3.2. Modification de la vue accueil\_view

Dans le contrôleur vous avez utilisé `$data['utilisateurs'] = $repository->findAll();`, dans votre vue la variable `$utilisateurs` est disponible, à vous de l'utiliser pour faire afficher la liste des description d'utilisateurs.

Vous devez maintenant avoir :



Chaque ancre devra être de la forme : `<a href="http://localhost/inf340tp/accueil/gallery/id"> description </a>`.

Pour vous aider, il faut savoir que : `$utilisateurs` est une collection d'objets, pour accéder à la propriété d'un objet la notation `->` est utilisée.

La méthode `site_url()` du helper `url` peut vous aider. Avant d'aller plus loin, il nous faut des images, nous allons passer au backoffice.

## 2.4. Le backoffice premier partie

Le backoffice est l'arrière boutique, la partie administration du site. Le lien login vous permet d'y accéder.

Le contrôleur `Accueil` du backoffice est un contrôleur sécurisé, il hérite de `MY_Controller` qui vérifie si la variable de session `logged_in` est positionnée, si elle ne l'est pas, le client est redirigé via le contrôleur `Sessions` vers un formulaire d'authentification. Si l'authentification réussie, la variable de session `logged_in` reçoit l'identifiant de l'utilisateur (`id`) qui vient de s'authentifier. Les variables de session sont visibles dans toutes les pages, ainsi le temps d'une session, l'utilisateur sera reconnu dans chaque page par son `id`.

Le code suivant permet d'accéder à la valeur de la variable `logged_in` : 

```
$varsession = $this->session->userdata('logged_in'); $id = $varsession['id'];
```

L'utilisateur non administrateur peut administrer sa galerie, faite afficher pour cet utilisateur la vue : `user_perso_view` du backoffice. La partie ajout d'image doit-être opérationnelle, elle est déjà fournie, nous pouvons retourner au frontoffice.

## 2.5. Le frontoffice suite et fin

Commençons par afficher la galerie d'un utilisateur.

### 2.5.1. Galerie utilisateur

Vous allez devoir en suivant les commentaires modifier la méthode `gallery($id)` du contrôleur `Accueil` pour afficher la vue du frontoffice `galerie_view`.

La view `galerie_view` est également à modifier.

Vous devez obtenir :



Pour vous aider, vous devez savoir que seul un utilisateur peut-être passé à la vue car chaque utilisateur possède la méthode `getImages()` qui permet d'accéder à ses images, les classes *proxy* ont oeuvré en ce sens. Chaque image est affichée sous forme miniature et est clickable, les URL sont de la forme : `http://localhost/inf340tp/accueil/display/id/url`. L'identifiant de l'utilisateur (`id`) est passé pour pouvoir revenir à sa galerie après la consultation d'une image, l'`url` permet de connaître l'image.

Un clic sur la miniature amène à la description de l'image.

## 2.5.2. Afficher une image et sa description

Pour afficher une image, vous aurez besoin de la méthode `display($id, $url)` du contrôleur `Accueil` et de la vue `image_view`.

Au final vous devez avoir :



Nous avons fini avec le frontoffice, nous pouvons revenir au backoffice.

## 2.6. Backoffice suite et fin

Nous avons laissé notre backoffice dans l'état suivant, les utilisateurs non administrateurs peuvent voir la page d'administration, la section "*mes images*" est opérationnelle, il nous faut maintenant gérer les utilisateurs en mode administrateur et gérer les données personnelles pour tous les modes.



### 2.6.1. L'administration des données personnelles

Nous allons maintenant ajouter la gestion des données personnelles, l'affichage est déjà réalisé par la méthode `index` du contrôleur `Accueil` du `backoffice` et la vue `user_perso_view` du `backoffice`.

Pour la modification, vous aurez à utiliser le contrôleur `Utilisateur` du `backoffice`, la méthode `updateOk()`, le dépôt `UtilisateurRepository` avec la méthode `updateUtilisateur($login, $password, $level, $description)` et la vue `update_success_view`.

Pour vous aider, vous pouvez vous inspirer du code `ImageRepository`, je vous conseil de tester à chacune de vos étapes.

### 2.6.2. L'administration des comptes

L'administration des comptes n'est accessible que pour les comptes administrateurs (level 0). Elle permet de créer et de supprimer des utilisateurs, un utilisateur ne peut se supprimer lui même.

#### 2.6.2.1. La page d'administration pour un administrateur

Commençons par modifier le contrôleur `Accueil` du `backoffice` pour faire afficher les vues : `user_perso_view` et `user_admin_view` pour les utilisateurs de niveau zéro.



### 2.6.2.2. La suppression d'un utilisateur

Pour supprimer un utilisateur vous aurez besoin du contrôleur `Utilisateur` du backoffice et de la méthode `delete($id)` qui utilise la méthode `delete($id)` du dépôt `UtilisateurRepository`.

Il vous faudra faire attention à empêcher un utilisateur de se supprimer lui-même en affichant la vue : `delete_error_view`.

### 2.6.2.3. L'ajout d'un utilisateur passe

La création d'un utilisateur passe par un formulaire de saisie, la vue `add_user_view`. Le contrôleur `Utilisateur` sollicite la vue via la méthode `create()` et modifie le modèle dans `createOk()` en utilisant la méthode `create($login, $level, $password, $description)` du dépôt `UtilisateurRepository`.

Nous avons ici fini avec le système d'information, vous allez maintenant passer en programmation où avec JavaScript vous allez traiter les données côté client.

Le temps restant doit vous permettre de commencer le devoir maison.

## 3. Devoir maison

Vous ne devez pas oublier de rendre votre devoir. En voici les contraintes :

- Le travail est réalisé par groupes de trois ou quatre, la soutenance est individuelle.
- Vous devez rendre votre travail avant le vendredi 4 novembre 2011 sur `ftp://ftp-exam.src/jberdjug/s3/inf340`, passé ce délai, le travail sera considéré comme non rendu.
- Vous devez avoir au moins deux tables liées par des contraintes d'intégrité.
- Vous devez rendre une archive unique portant vos noms contenant :

1. Le projet eclipse
2. Un fichier de création de la base, de tables et éventuellement des insertion et ceux si vous avez décidé de ne pas utiliser Doctrine.
3. Un fichier au format PDF portant vos noms contenant :
  - a. le sujet choisi ;
  - b. une notice de déploiement contenant toutes les informations que vous jugez utiles au déploiement : nom de l'alias, nom de la base, comptes et mots de passes, ....;
  - c. les cas d'utilisations;
  - d. selon l'approche choisie, un diagramme de classe UML ou un Modèle Logique de Données.

Les critères de l'évaluation sont les suivants :

- La robustesse du site : la vérification des formulaires, les tables vides, ... ;
- La qualité de la base de données ou du modèle objet : structure des tables, contraintes d'intégrités, ... ;
- La qualité de l'XHTML qui doit-être strict ;
- La CSS doit fonctionner sous tous types de navigateurs ;
- La pertinence des réponses posées pendant l'évaluation et la clarté de l'exposé.

---

# Annexe A. Création des modèles

Vous trouverez ici, une méthode pour créer vos modèles en utilisant une solution de reverse-ingeniering.

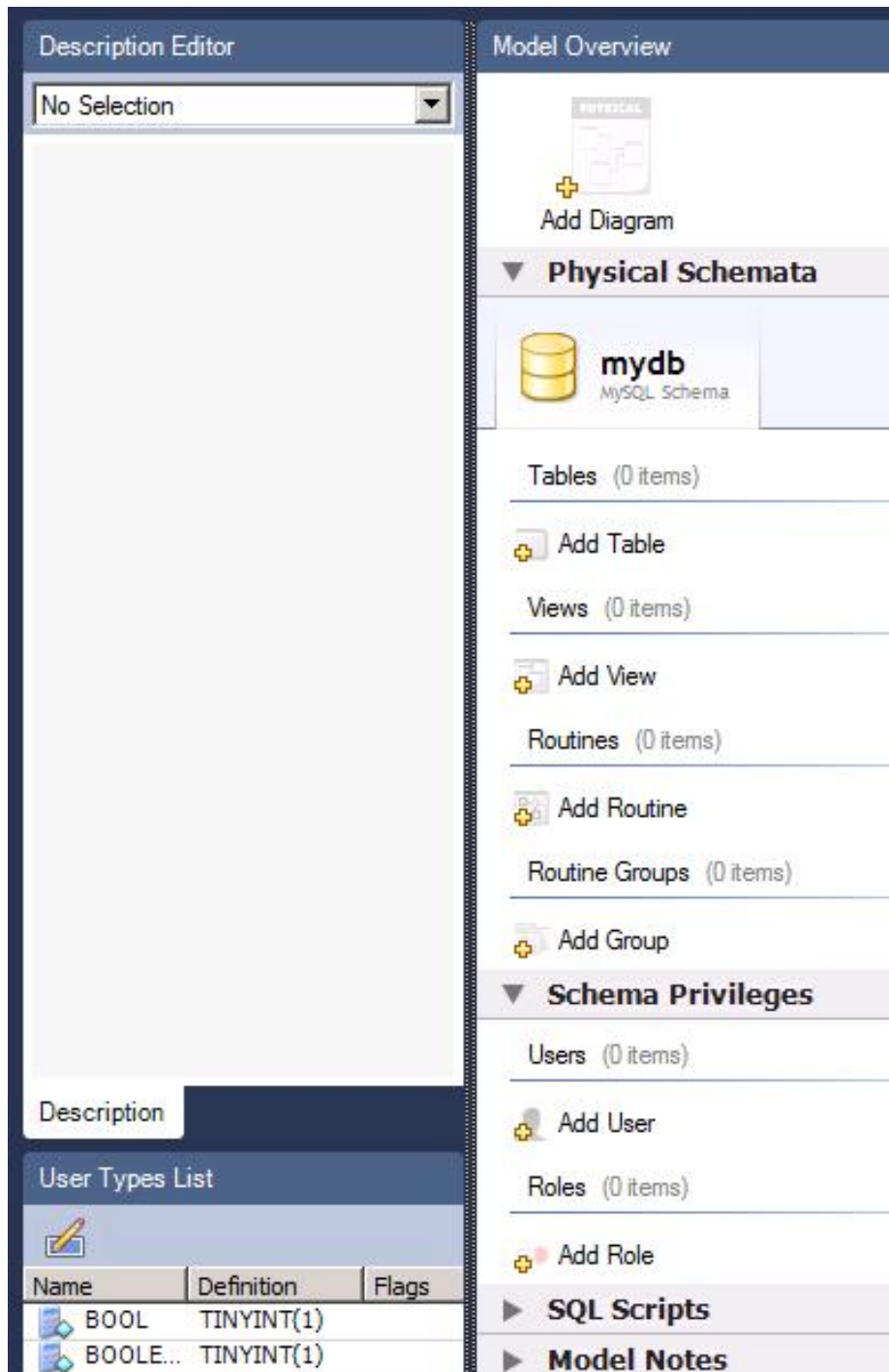
## 1. MySQL Workbench

MySQL Workbench est un système de conception de base de données sous forme graphique qui intègre la conception d'un MLD (Modèle logique de données) et la création d'un MPD (Modèle physique de données) :

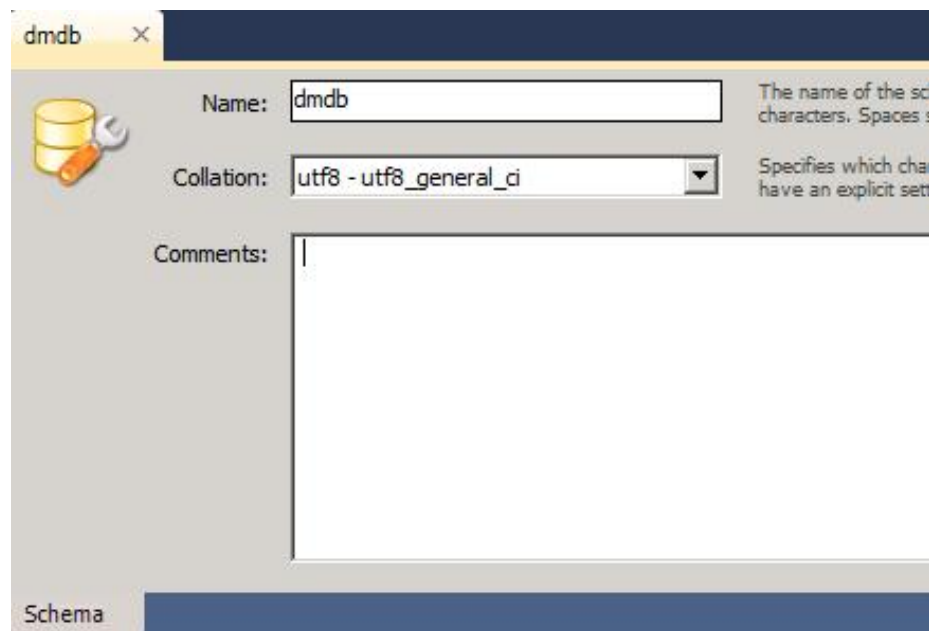
1. Lancer MySQL Workbench et choisir "create new EER Model".



Figure A.1. Création d'un modèle Entité Relationnel



2. Éditer votre schéma pour le renommer en *dmdb* et choisir l'encodage UTF-8.

**Figure A.2. Le choix du nom et de l'encodage du schéma**

3. Ajouter deux tables *article* et *image*

**Figure A.3. Création de deux tables vides**

4. Spécifier les colonnes des tables

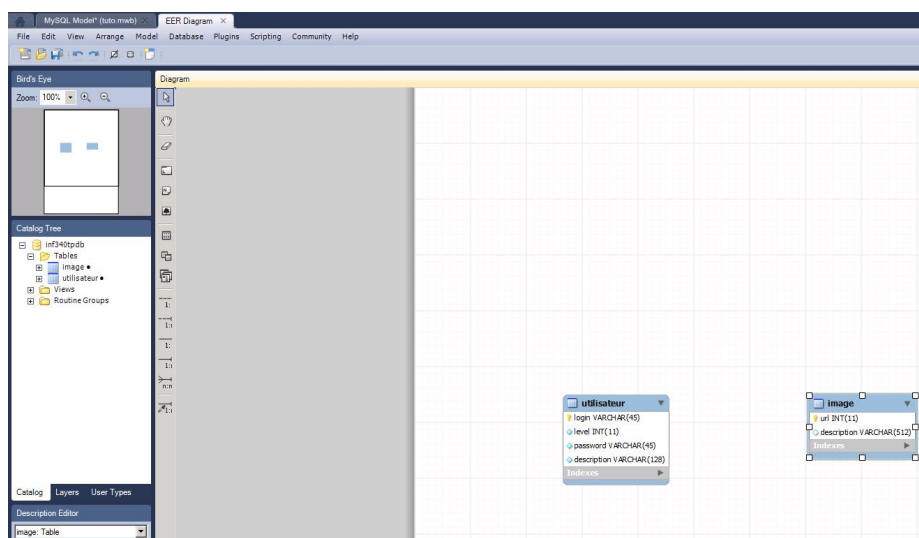
**Figure A.4. La table utilisateur**

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
login	VARCHAR(45)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
level	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
password	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
description	VARCHAR(128)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

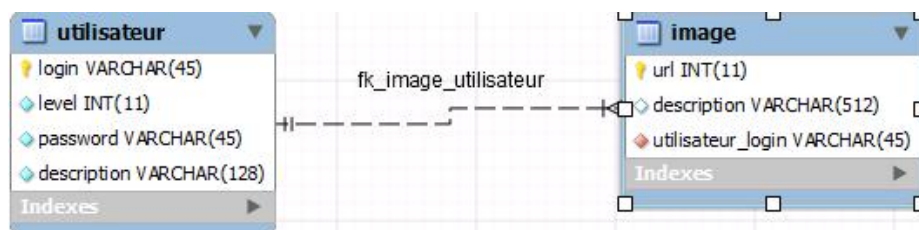
**Figure A.5. La table image**

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
url	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
description	VARCHAR(512)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

5. Créer un nouveau diagramme et y placer les tables

**Figure A.6. La conception du diagramme**

6. Création d'un association 1-n non identifiée

**Figure A.7. Création d'une association 1-n**

7. Renommage des champs créer et amélioration de la contrainte d'intégrité référentielle.

**Figure A.8. Renommage des champs générés**

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
url	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
description	VARCHAR(512)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
utilisateur	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**Figure A.9. Modification d'une contrainte d'intégrité référentielle**

Foreign Key Name	Referenced Table	Column	Referenced Column	Foreign Key Options
fk_image_utilisateur	inf340tpdb.`utilisateur`	description	login	On Update: CASCADE On Delete: CASCADE

8. Génération du script SQL (Export)

```
SET @OLD_UNIQUE_CHECKS=@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@SQL_MODE, SQL_MODE='TRADITIONAL';
```

```
-----
-- Table `dmdb`.`utilisateur`
-----
```

```

DROP TABLE IF EXISTS `dmdb`.`utilisateur` ;

CREATE TABLE IF NOT EXISTS `dmdb`.`utilisateur` (
  `login` VARCHAR(45) NOT NULL ,
  `level` INT(11) NOT NULL ,
  `password` VARCHAR(45) NOT NULL ,
  `description` VARCHAR(128) NOT NULL ,
  PRIMARY KEY (`login`) ,
  UNIQUE INDEX `UNIQ_1D1C63B36DE44026` (`description` ASC) )
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

-----
-- Table `dmdb`.`image`
-----
DROP TABLE IF EXISTS `dmdb`.`image` ;

CREATE TABLE IF NOT EXISTS `dmdb`.`image` (
  `url` INT(11) NOT NULL AUTO_INCREMENT ,
  `description` VARCHAR(512) NULL DEFAULT NULL ,
  `utilisateur` VARCHAR(45) NOT NULL ,
  PRIMARY KEY (`url`) ,
  INDEX `fk_image_utilisateur` (`utilisateur` ASC) ,
  CONSTRAINT `fk_image_utilisateur`
    FOREIGN KEY (`utilisateur`)
      REFERENCES `dmdb`.`utilisateur` (`login` )
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```

## 2. PHPMYAdmin

Nous allons maintenant utiliser le script de création pour construire notre base de données et ses deux tables.

Dans *PHPMYAdmin* créer une base de données nommé *dmdb* avec comme encodage UTF-8, un utilisateur avec tout pouvoir sur cette base. Dans cette démonstration l'utilisateur est *admdmdb* et n'a pas de mot de passe.

Figure A.10. Un utilisateur et une base

The screenshot shows the 'Add a new User' interface in phpMyAdmin. The user is being added to the 'localhost' server. The 'User name' is 'admdmdb' and the 'Host' is 'localhost'. The password is currently empty. The 'Database for user' section has 'None' selected. The 'Global privileges' section is expanded, showing four categories: Data, Structure, Administration, and Resource limits. All checkboxes in these categories are checked. The 'Data' category includes SELECT, INSERT, UPDATE, DELETE, and FILE. The 'Structure' category includes CREATE, ALTER, INDEX, DROP, CREATE TEMPORARY TABLES, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, EXECUTE, CREATE VIEW, EVENT, and TRIGGER. The 'Administration' category includes GRANT, SUPER, PROCESS, RELOAD, SHUTDOWN, SHOW DATABASES, LOCK TABLES, REFERENCES, REPLICATION CLIENT, REPLICATION SLAVE, and CREATE USER. The 'Resource limits' section has four input fields, all set to 0: MAX QUERIES PER HOUR, MAX UPDATES PER HOUR, MAX CONNECTIONS PER HOUR, and MAX USER\_CONNECTIONS.

Importer dans la base les tables créée en utilisant le script SQL précédemment généré.

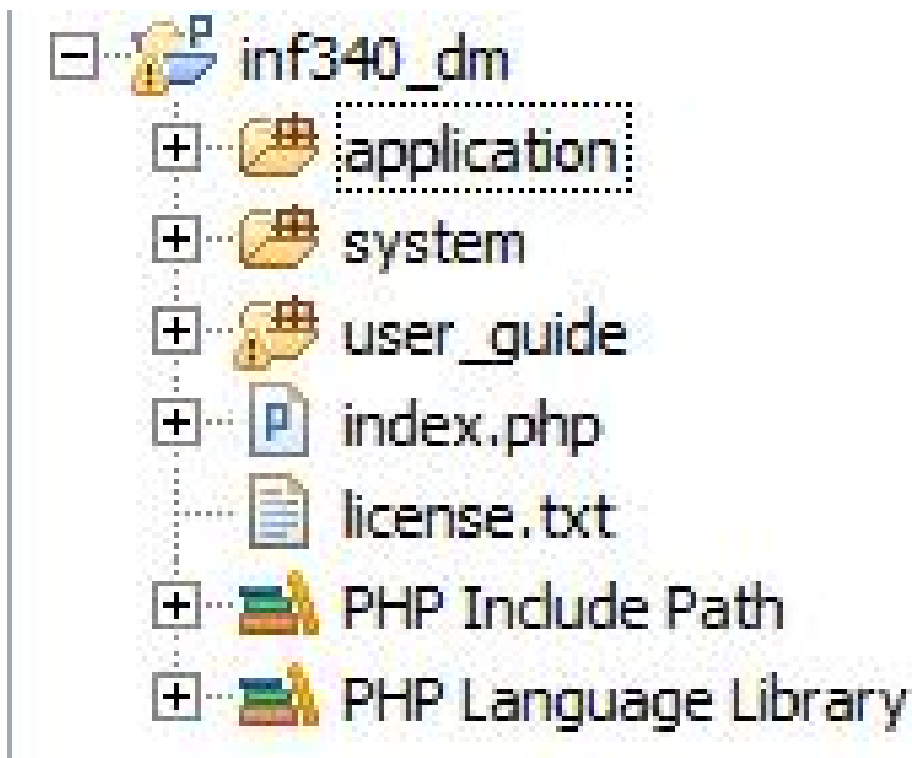
## 3. CodeIgniter et Doctrine

Nous allons maintenant créer nos modèles doctrine depuis une base existante

### 3.1. Import du projet vide dans le workspace

Importer le projet vide fourni, vous devez obtenir:

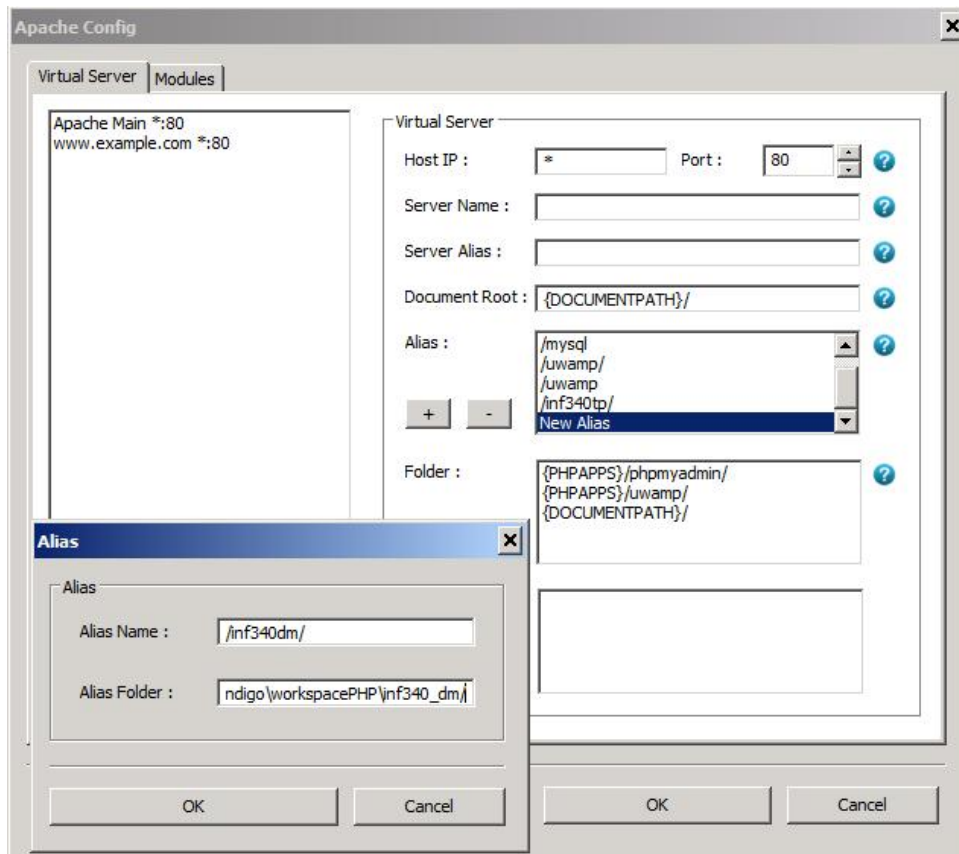
Figure A.11. Import du projet



### 3.2. Création de l'alias

Créer un alias nommé `inf340dm` qui référence la racine de votre projet :

Figure A.12. Création de l'alias



### 3.3. Configuration de CodeIgniter

Dans CodeIgniter, vous devrez :

1. Modifier le fichier `.htaccess` à la racine du projet :

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule ^(.*)$ /inf340dm/index.php/$1 [L]
</IfModule>
<IfModule !mod_rewrite.c>
  ErrorDocument 404 /inf340dm/index.php
</IfModule>
Options +MultiViews
```

2. Modifier `application/config/config.php` pour indiquer l'URL de votre site :

```
$config['base_url'] = 'http://localhost/inf340dm/';
```

3. Modifier `application/config/database.php` pour indiquer le serveur, la base et l'utilisateur :

```
$db['default']['hostname'] = 'localhost';
$db['default']['username'] = 'admdmdb';
$db['default']['password'] = '';
$db['default']['database'] = 'dmdb';
$db['default']['dbdriver'] = 'mysql';
$db['default']['dbprefix'] = '';
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = '';
```

```

$db['default']['char_set'] = 'utf8';
$db['default']['dbcollat'] = 'utf8_general_ci';
$db['default']['swap_pre'] = '';
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;

```

### 3.4. Le reverse Engineering

Vous pouvez maintenant utiliser le script <http://localhost/inf340dm/install/createModelFromDB>. Ce script a créé deux entités depuis la base de données:

```

/**
 * Utilisateur
 *
 * @Table(name="utilisateur")
 * @Entity
 */
class Utilisateur
{
    /**
     * @var string $login
     *
     * @Column(name="login", type="string", length=45, nullable=false)
     * @Id
     * @GeneratedValue(strategy="IDENTITY")
     */
    private $login;

    /**
     * @var integer $level
     *
     * @Column(name="level", type="integer", nullable=false)
     */
    private $level;

    /**
     * @var string $password
     *
     * @Column(name="password", type="string", length=45, nullable=false)
     */
    private $password;

    /**
     * @var string $description
     *
     * @Column(name="description", type="string", length=128, nullable=false)
     */
    private $description;
}

```

```

<?php

use Doctrine\ORM\Mapping as ORM;

/**
 * Image
 *
 * @Table(name="image")
 * @Entity
 */
class Image
{
    /**
     * @var integer $url
     *

```



```

    * @Column(name="url", type="integer", nullable=false)
    * @Id
    * @GeneratedValue(strategy="IDENTITY")
    */
    private $url;

    /**
     * @var string $description
     *
     * @Column(name="description", type="string", length=512, nullable=true)
     */
    private $description;

    /**
     * @var Utilisateur
     *
     * @ManyToOne(targetEntity="Utilisateur")
     * @JoinColumn({
     *     @JoinColumn(name="utilisateur", referencedColumnName="login")
     * })
     */
    private $utilisateur;
}

```

### 3.5. Création des dépôts

Nous allons regrouper les méthodes d'accès aux entités dans des classes, les dépôts. Un dépôt hérite de `EntityRepository` et dispose donc des méthodes comme `find`.

```

<?php

namespace models\repositories;

class UtilisateurRepository extends \Doctrine\ORM\EntityRepository {

}

?>

```

```

<?php

namespace models\repositories;

class ImageRepository extends \Doctrine\ORM\EntityRepository {

}

?>

```

### 3.6. La modification des modèles

Nos entités ne sont pas complètes et contiennent des erreurs.

1. Ajout de l'espace de nommage, dans chaque classe il vous faut spécifier l'espace de nommage.

```
namespace models;
```

2. Correction des champs auto-incrémentant. Toutes les clés sont générées en auto-incrémentant or pour utilisateur, ce n'est pas le cas.

```
/**
 * @var string $login

```

```
*
* @Column(name="login", type="string", length=45, nullable=false)
* @Id
* @GeneratedValue(strategy="NONE")
*/
```

3. Spécification des dépôts, une entités doit pouvoir connaître son dépôt.

```
@Entity (repositoryClass="\models\repositories\UtilisateurRepository")
```

```
@Entity(repositoryClass="\models\repositories\ImageRepository")
```

4. Ajout des contraintes d'unicité.

```
@Table(name="utilisateur",
        uniqueConstraints={@UniqueConstraint(columns={"description"})})
```

5. Modification des contraintes d'intégrité référentielles pour rajouter le onDelete Cascade.

```
/**
 * @var Utilisateur
 *
 * @ManyToOne(targetEntity="Utilisateur")
 * @JoinColumns({
 *     @JoinColumn(name="utilisateur",
 *                  referencedColumnName="login", onDelete="Cascade")
 * })
 */
```

La contrainte onUpdate Cascade n'est pas encore supportée par doctrine.

## 3.7. Génération de la base à partir du modèle

Il nous faut commencer par modifier le script <http://localhost/inf340dm/install/install> pour spécifier les tables que nous souhaitons créer:

```
$classes = array(
    $em->getClassMetadata('\models\Utilisateur'),
    $em->getClassMetadata('\models\Image')
);
```

Nous pouvons maintenant régénérer notre base de données depuis notre modèle et étudier le SQL obtenu en utilisant le script.

```
CREATE TABLE IF NOT EXISTS `image` (
  `url` int(11) NOT NULL AUTO_INCREMENT,
  `utilisateur` varchar(45) DEFAULT NULL,
  `description` varchar(512) DEFAULT NULL,
  PRIMARY KEY (`url`),
  KEY `IDX_C53D045F1D1C63B3` (`utilisateur`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

CREATE TABLE IF NOT EXISTS `utilisateur` (
  `login` varchar(45) NOT NULL,
  `level` int(11) NOT NULL,
  `password` varchar(45) NOT NULL,
  `description` varchar(128) NOT NULL,
  PRIMARY KEY (`login`),
  UNIQUE KEY `UNIQ_1D1C63B36DE44026` (`description`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `image`
  ADD CONSTRAINT `FK_C53D045F1D1C63B3` FOREIGN KEY (`utilisateur`)
    REFERENCES `utilisateur` (`login`) ON DELETE CASCADE;
```

## 3.8. Amélioration des entités

Les entités ont été améliorées en ajoutant des constructeurs et des assesseurs. Mais aussi des

### règles de visibilité

Un utilisateur souhaite voir ses images

```
/**
 * @OneToMany(targetEntity="Image", mappedBy="utilisateur")
 */
private $images;
```

et dans le constructeur

```
$this->images=new ArrayCollection;
```

### Un cycle de vie

Avant de supprimer une image nous supprimons les fichiers.

```
/**
 * Image
 *
 * @Table(name="image")
 * @Entity(repositoryClass="\models\repositories\ImageRepository")
 * @HasLifecycleCallbacks
 */
```

```
/**
 * @preRemove
 */
public function deleteFile()
{
}
```