Système d'information

Développement d'une application Web

Jean-François Berdjugin
Vincent Lestideau

Système d'information: Développement d'une application Web

par Jean-François Berdjugin et Vincent Lestideau

Date de publication Septembre 2011

Résumé

Nous allons construire, une application Web proposant la vente de tee-shirts. Notre application repose sur une architecture trois tiers[1] ; 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 primitives MySQL puis PDO (PHP Data Objects). En TP nous utiliserons le framework CodeIgniter et l'ORM (Object-Relational Mapping) Doctrine.

[1] Un tiers est une couche.

Table des matières

Liste des illustrations

Liste des exemples

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 le 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.

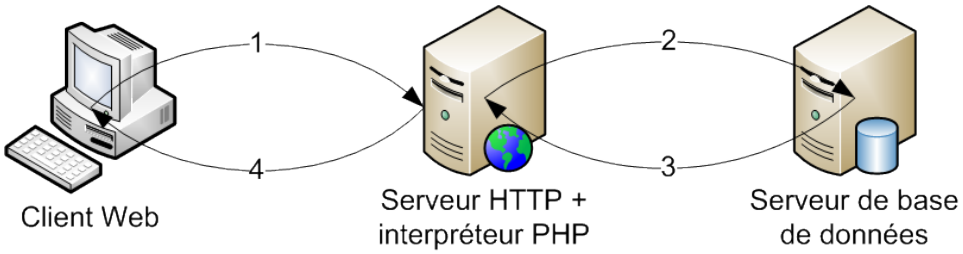
Chapitre 2. Présentation de l'architecture et des outils

Le développement Web repose sur une architecture distribuées, 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[1]. 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 soustraite 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 par directement mais nous utiliserons l'interface "graphique" de WAMP et

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

des fichiers `.htaccess`. Les fichiers `.htaccess` sont des fichiers de configuration des serveurs Web Apache 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`[2].

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" (`e:`) un répertoire nommé `workspacePHP`, 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. A la racine du projet, au même niveau que le fichier `index.php` vous aller créer un fichier nommé `test.php`, et y placer 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>Pas interprété
    <?php phpinfo();?>
    </p>
  </body>
 </html>
```

5. Créer dans apache un alias nommé inf340td pointant vers le répertoire physique `E:\workspacePHP\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.*

6. Tester depuis votre client Web à l'URL : `http://localhost/inf340td/test.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

[2]Uniquement depuis PHP 5.3 et en CGI et FastCGI.

```
<?php ?>
```

. `phpinfo();` est une fonction qui donne des informations sur la configuration courante de PHP.

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 5.3 la plus récente[3].

Une fois, la version 5.3 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.[4].

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 peu

  vent 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 : PHPMyAdmin.

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

[3] Il n'est possible qu'il n'y ai qu'une version d'installée.
[4] Dans le `php.ini` *;* marque une ligne de commentaire

Lors de la suite du développement vous pourrez utiliser PHPMyAdmin pour vérifier le résultat de vos scripts. *Nous avons choisi d'utiliser le compte root de MySQL, pour des raisons de sécurité ce ne sera jamais le cas en production.*

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 a 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 (__conbstruct), 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 au 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` et égale à la valeur de `$b`,

$a === $b
: est vraie si la valeur de `$a` et 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 peut les mêmes structure qu'en *java* , il existe cependant des écritures alternatives comme par exemple pour la conditionnelle :

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

Je vous conseil de ne pas utiliser les écritures 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 $a. "est plus petit ou egale à".$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'incrémentation
                   (post-incrémentation)  */
}
```

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

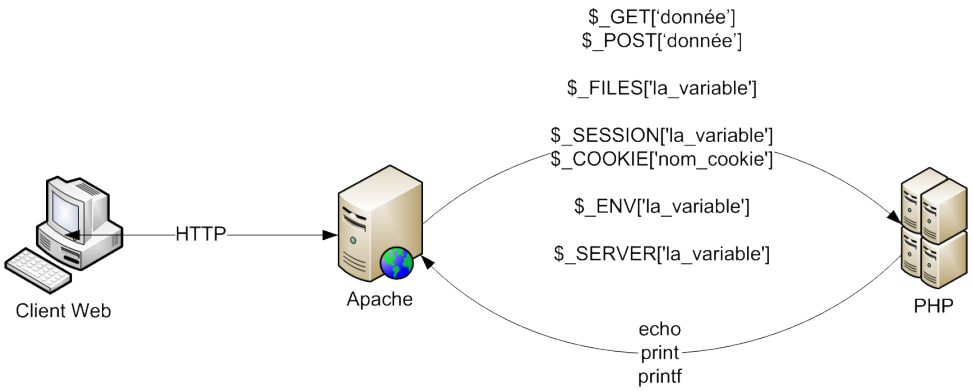
```
$arr = array("un", "deux", "trois");
foreach ($arr as $key => $value) {
    echo "Clé : $key; Valeur : $value<br />\n";
}
/*
Clé : 0; Valeur : un
Clé : 1; Valeur : deux
Clé : 2; Valeur : trois
 */

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

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

2. La communication entre le serveur Web et PHP

Figure 3.1. Communication entre le serveur Web et PHP

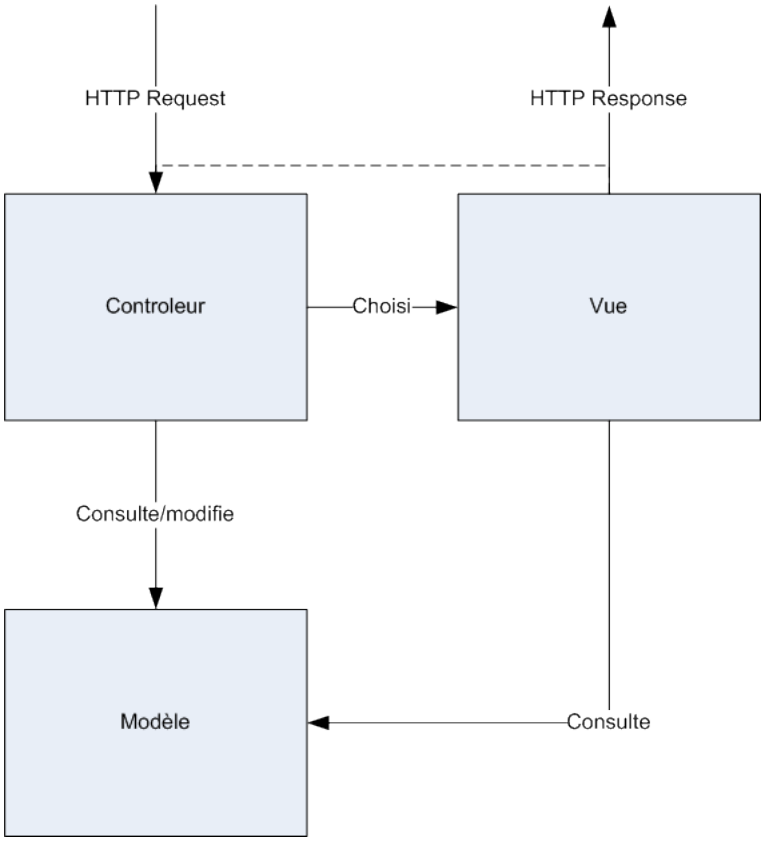


Le serveur Web et PHP communique, PHP reçoit des informations du serveur Web qui lui même en tient une partie du client. Pour ce qui nous intéresse ces informations sont dans les tableaux associatifs : $_GET, $_POST et $_REQUEST (un tableau associatif qui contient par défaut le contenu des variables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

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 3.2. MVC en PHP



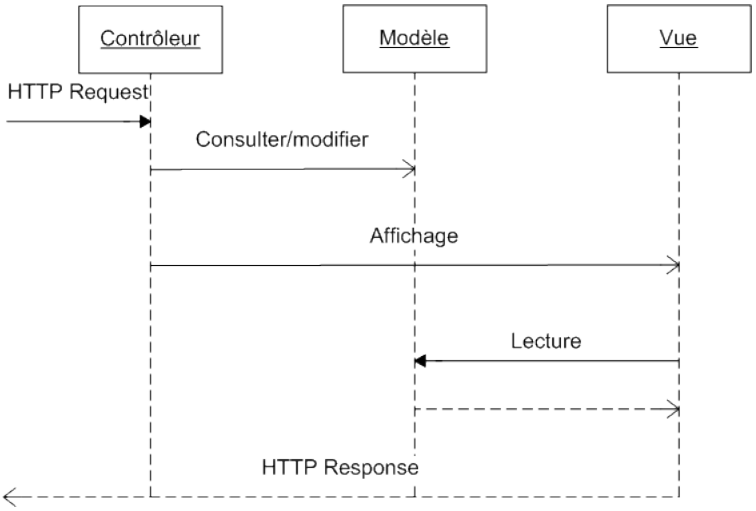
Séparer les concepts permets de séparer les métiers et faciliter la réutilisation. Dans ce patron de conception nous avons :

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 défini 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 3.3. 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 de 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 ou 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`.

Vous devez :

1. Définir la constante `APPPATH` pour quelle indique le répertoire `application`. Vous disposez déjà de la constante `FCPATH`, le point (.) réalise la concaténation et la constante prédéfinie `DIRECTORY_SEPARATOR` représente le séparateur de répertoire (\ sous Windows, / sous Linux). Si `FCPATH` vaut `D:\workspaces\workspacePHP\inf340_td\` alors `APPPATH` devra valoir `D:\workspaces\workspacePHP\inf340_td\application\`. Pour rappel, un echo vous permet de réaliser un affichage.

2. Inclure le fichier de constantes `constants.php` du répertoire `config` d'`application`*Il vous faut préférer require_once ou require aux includes, ainsi un fichier non trouvé provoque l'arrêt du script*.

3. Le fichier `constants.php` permet de définir l'URL du site, sa structure (`models`, `views`, `controllers`) et la connexion à la base de données (`DB_HOST`, `DB_NAME`, `DB_LOGIN`, DB_PASSWORD), vous n'avez qu'a modifier l'URL.

4. Inclure le contrôleur de page utilisateur.php*Nous sommes ici dans un cas simple et nous n'avons pas à trier les contrôleurs de page, nous n'en n'avons qu'un.*.

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 une utilisateur choisi.

updateOk
: qui permet de modifier l'utilisateur choisi.

Nous allons progressivement implémenter ces actions mais avant vous devez faire en sorte que :

- l'URL http://localhost/inf340td/index.php?action=create affiche create,
- l'URL http://localhost/inf340td/index.php?action=read affiche read,
- l'URL http://localhost/inf340td/index.php?action=update affiche update,
- l'URL http://localhost/inf340td/index.php?action=updateOk affiche updateOk,
- l'URL http://localhost/inf340td/index.php?action=delete affiche deleteOK.

Un contrôleur contrôle la logique de navigation, nous allons maintenant ajouter les fonctionnalités.

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.

4.3.1. La connexion et la déconnexion

Nous utilisons toujours la même base et pour chaque requête, nous avons choisi de nous connecter d'effectuer la requête et de nous déconnecter. En utilisant la documentation de `http://www.php.net/manual/fr/function.mysql-connect.php`, de `http://www.php.net/manual/fr/function.mysql-select-db.php` et de `http://www.php.net/manual/fr/function.mysql-close.php`, compléter le code de `connect()` et `disconnect()` de `db_utils.php`.

4.3.2. Le modèle

La méthode getAll() de `utilisateur_utils.php` est déjà fournie, en utilisant la documentation de MySQL essayer de comprendre son fonctionnement.

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, ici accueil_view.php.

A vous de réaliser les opérations précédemment décrites[1].

[1] Souvent les pages partages la même css et le même JS, c'est pourquoi on définit des templates.

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.

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`.

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/inde.php?create`, sa méthode est *post* et l'information envoyée est *identifiant* et *mot_de_passe*.

4.4.1. Le modèle

Compléter la méthode `create($identifiant, $mot_de_passe)` du modèle `utilisateur_utils.php` pour ajouter un nouvel utilisateur.

4.4.2. Le contrôleur

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'acceuil.

Pour réafficher la page d'accueil vous aller informer le client web que `http://localhost/inf340td/inde.php?create` n'est pas disponible mais que `http://localhost/inf340td/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 chaqu'une des ancres contienne comme href `http://localhost/inf340td/index.php?action=delete&identifiant=valeur` où valeur est un identifiant. Faites ce premier travail et vérifiez l' HTML obtenu.

4.5.1. Le modèle

Compléter la méthode `delete($identifiant)` du modèle `utilisateur_utils.php` 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`.

4.6. 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/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.

update($identifiant, $mot_de_passe)
: qui modifie le mot de passe de l'utilisateur correspondant à l'identifiant.

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 précedement sélectionné. L'action du formlaire est `http://localhost/inf340td/index.php?action=updateOk`. Le champ identifiant est en `readonly`.

4.6.5. La méthode updateOk du contôleur

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 redirigevers 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 les modèles fournis avec CodeIgniter mais nous utiliserons L'ORM Doctrine est 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 suffirons, 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://www.doctrine-project.org/projects/orm/2.0/docs/en` et plus particulièrement de la partie référence. Pour CodeIgniter, vous pourrez utiliser la documentation fournie avec le projet.

1. Mise en place

Nous aurons comme en TD a 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. 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. 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
```

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 *librairies* et "*helpers*" de CodeIgniter, les librairies 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 `ressources` non fourni avec CodeIgniter contient les ressources du site (images, CSS, ...).

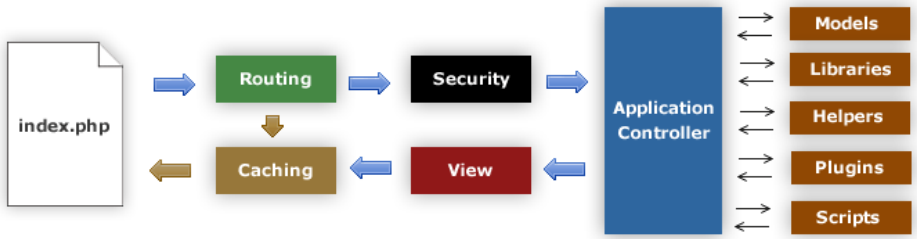
Dans le répertoire application vous trouverez un répertoire pour les contrôleurs (`contollers`), un pour les modèles (`models`) et enfin un pour les vues (`views`).

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



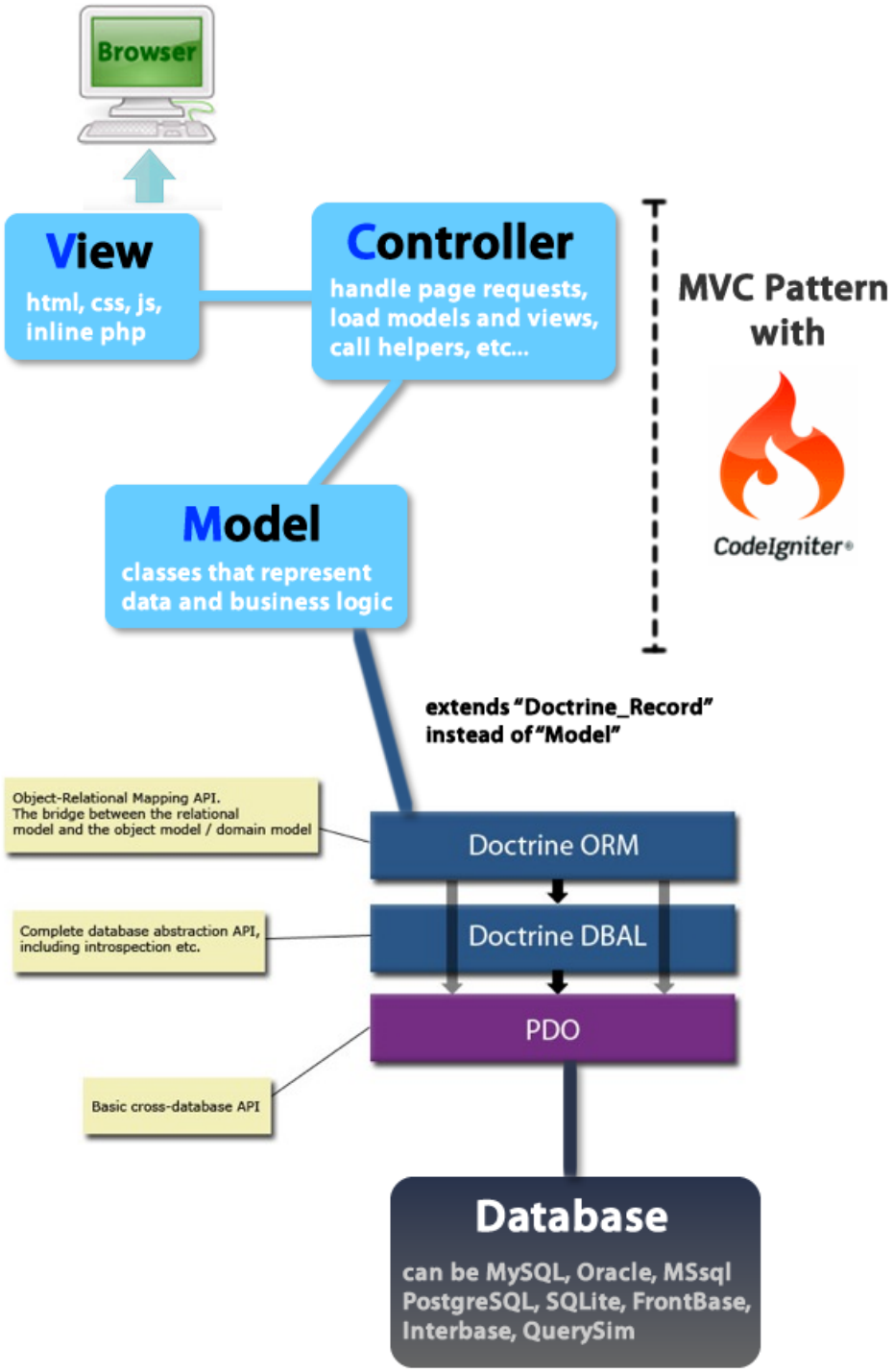
2.2. Notre modèle

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 `repositeries` 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*Sous WAMP mais pas sous uWamp qui ne contient pas l'extension PHP mcrypt.*, vous pouvez regarder la documentation de CodeIgniter pour en comprendre le fonctionnement.

2.2.2. Création de nos table

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` (
  `login` varchar(45) NOT NULL,
  `level` int(11) DEFAULT 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;
```

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 $em->persist($utilisateur1) 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.

Vous devez avoir l'affichage suivant :

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, 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 $utlisateurs est disponible, à vous de l'utiliser pour faire afficher le liste des description d'utilisateurs.

Vous devez maintenant avoir :

Chaque ancre devra être de la forme : description .

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 position une variable de session `loggedin` si l'authentification réussi.

Le code suivant permet d'accéder à la valeur de la variable `loggedin` : $varsession = $this->session->userdata('loggedin'); $login = $varsession['login'];

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, vous pouvons retourner au frontoffice.

2.5. Le frontoffice suite et fin

Commençons par affiche la galerie d'un utilisateur.

2.5.1. Galerie utilisateur

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

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/login/url`.

Un clic 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($login, $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 avions laissé notre backoffice dans l'état suivant, les utilisateurs non administrateurs peuvent voir la page d'administration, la section "*mes images*" est opérationnelle.

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 chaque étape.

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($login)` qui utilise la méthode `delete($login)` du dépôt `UtilisateurRepository`.

Il vous faudra faire attention a 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 se 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 coté 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 décidez 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'allias, 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

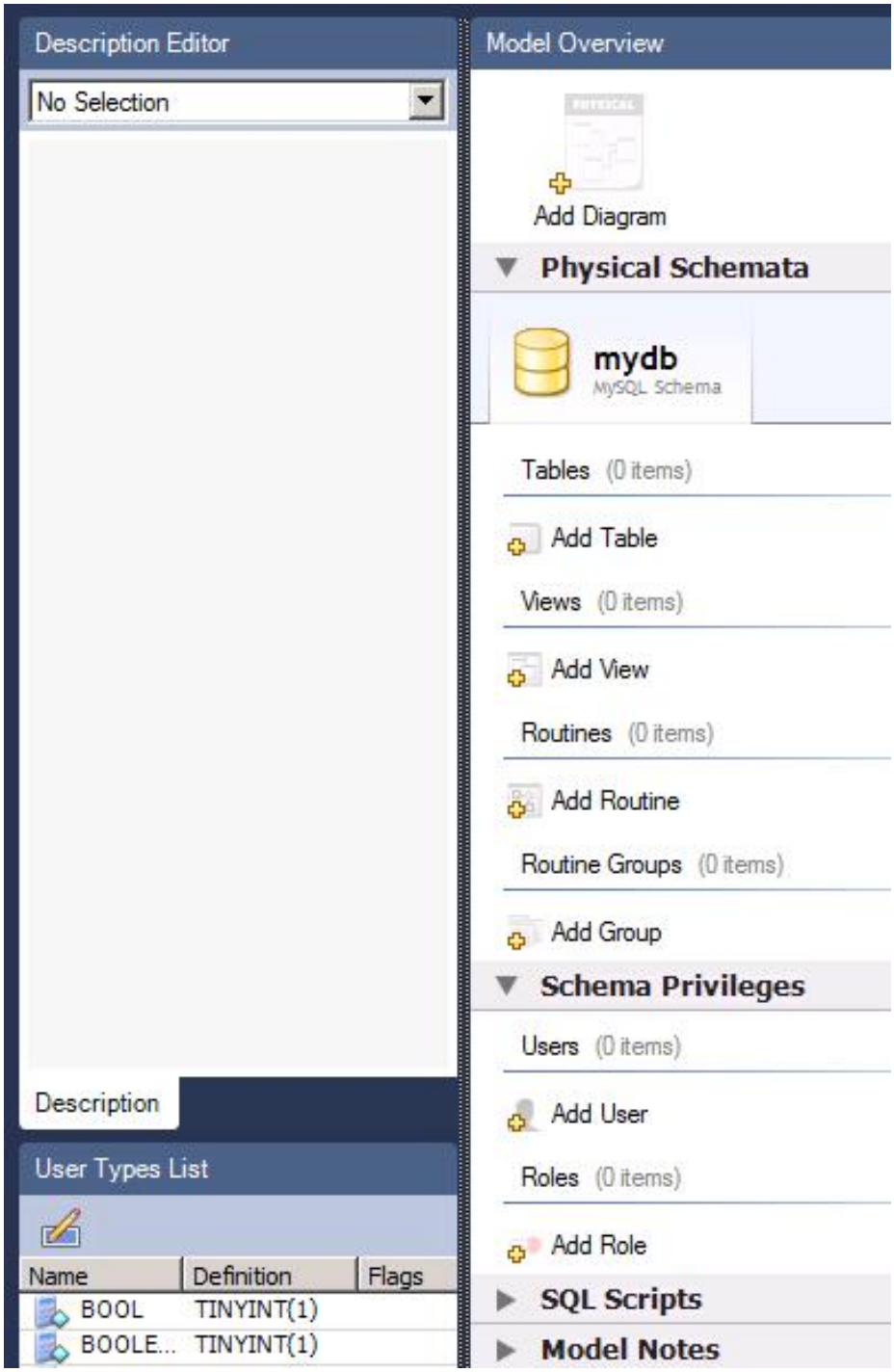
Vous trouverez ici, une méthode pour créer vos modèles en utilisant une solution de reverse-ingeniering.

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) :

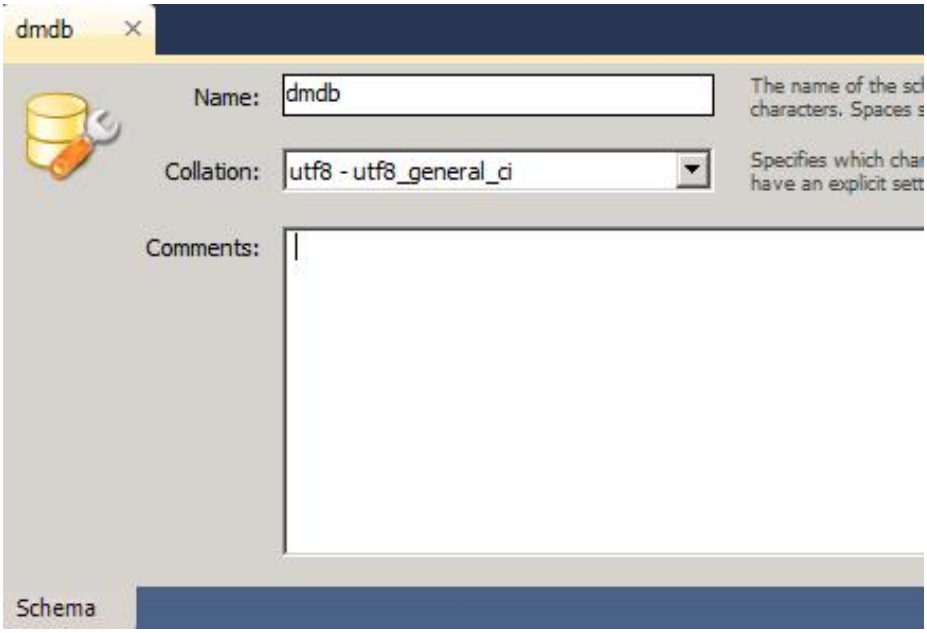
1. Lancer MySQL Workbench et choisir "create new EER Model".

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

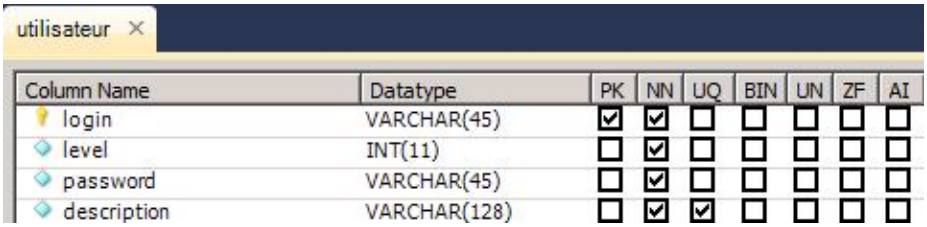
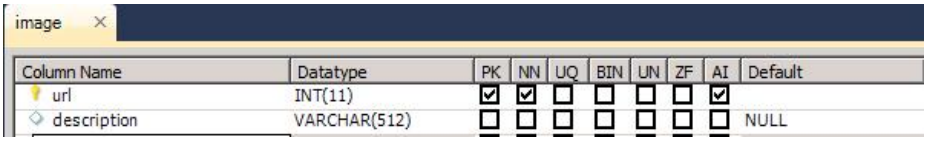
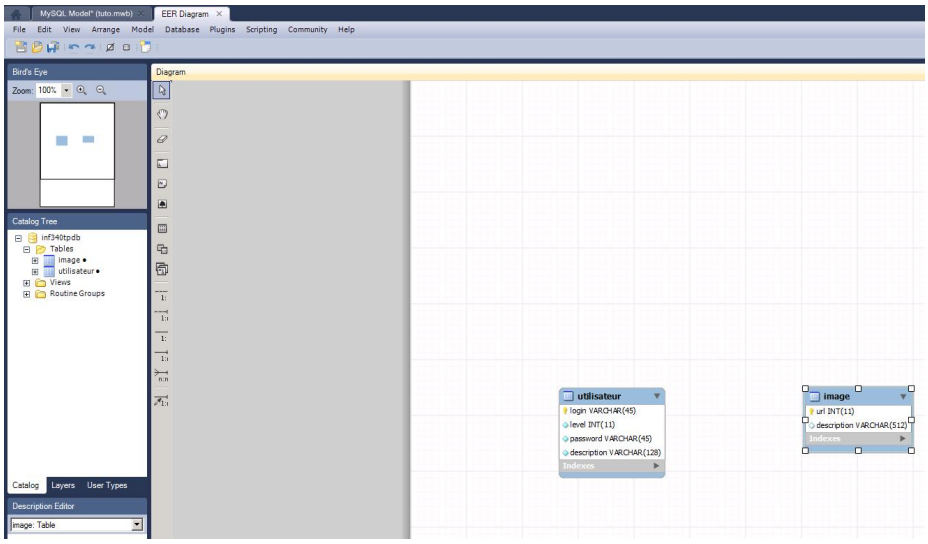


Figure A.5. La table image



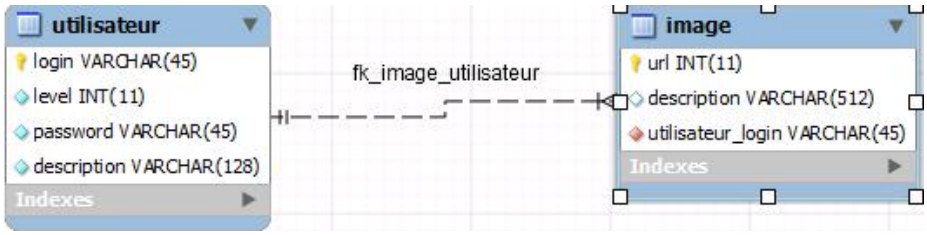
5. Créer un nouveau diagramme et y placer les tables

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

Figure A.8. Renommage des champs générés

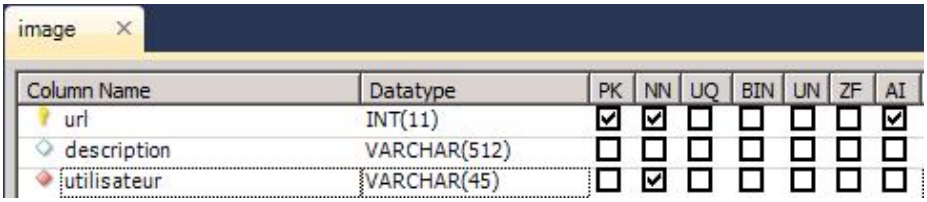
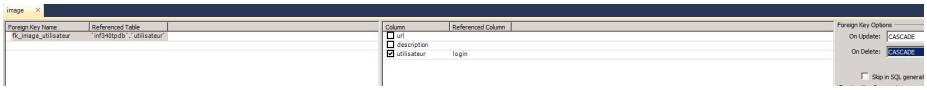


Figure A.9. Modification d'une contrainte d'intégrité référentielle



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`
-- -----------------------------------------------------
```

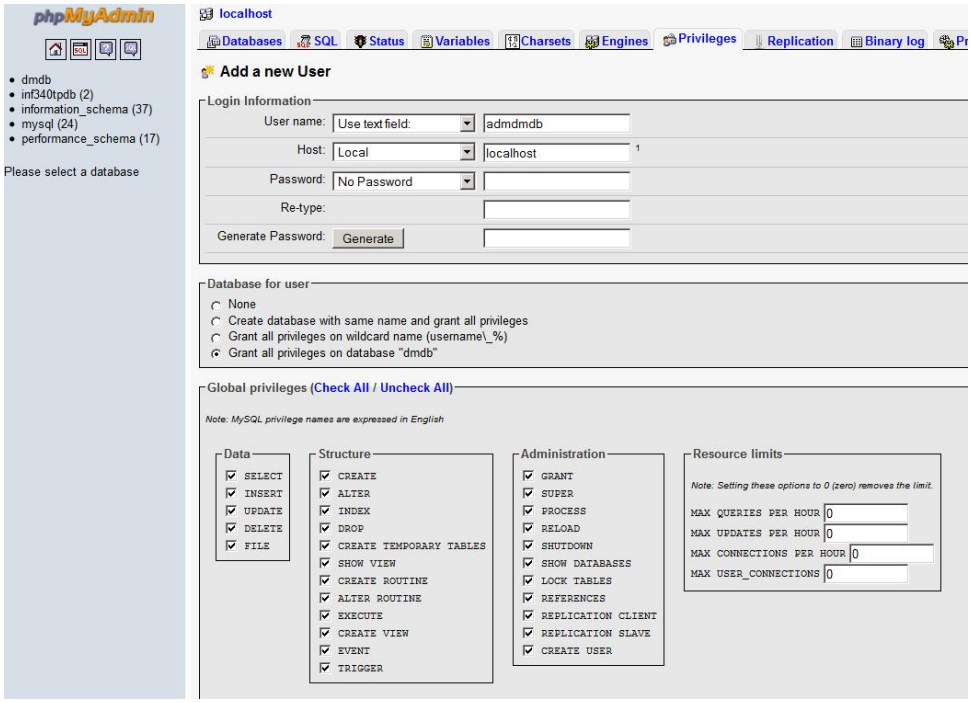
```
DROP TABLE IF EXISTS `dmdb`.`utilisateur` ;

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 donnée 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



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

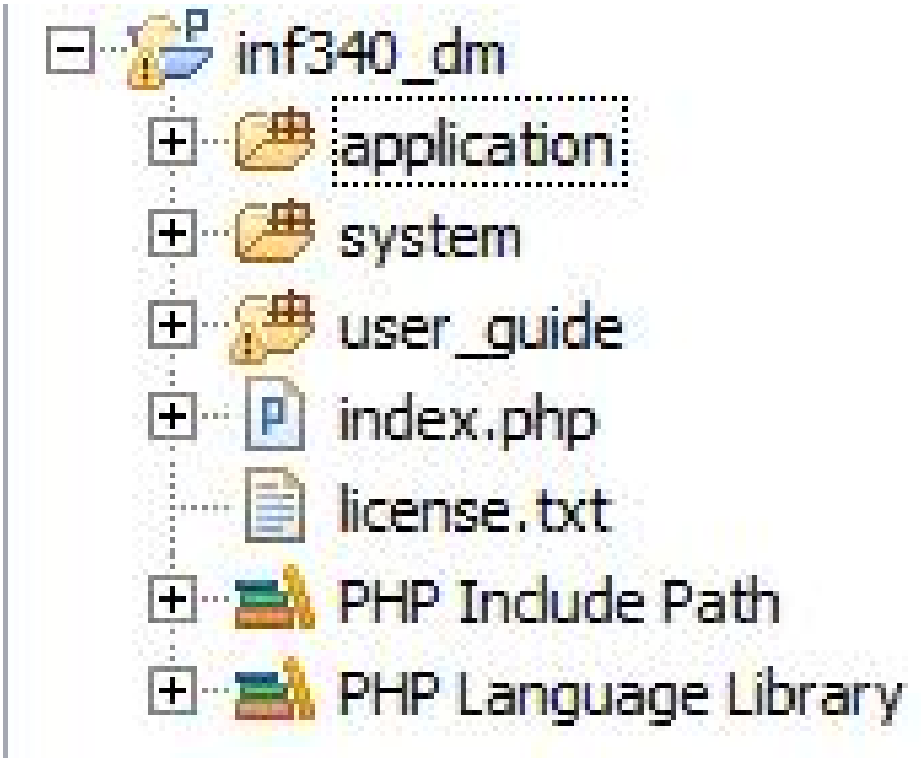
3. CodeIgniter et Doctrine

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:

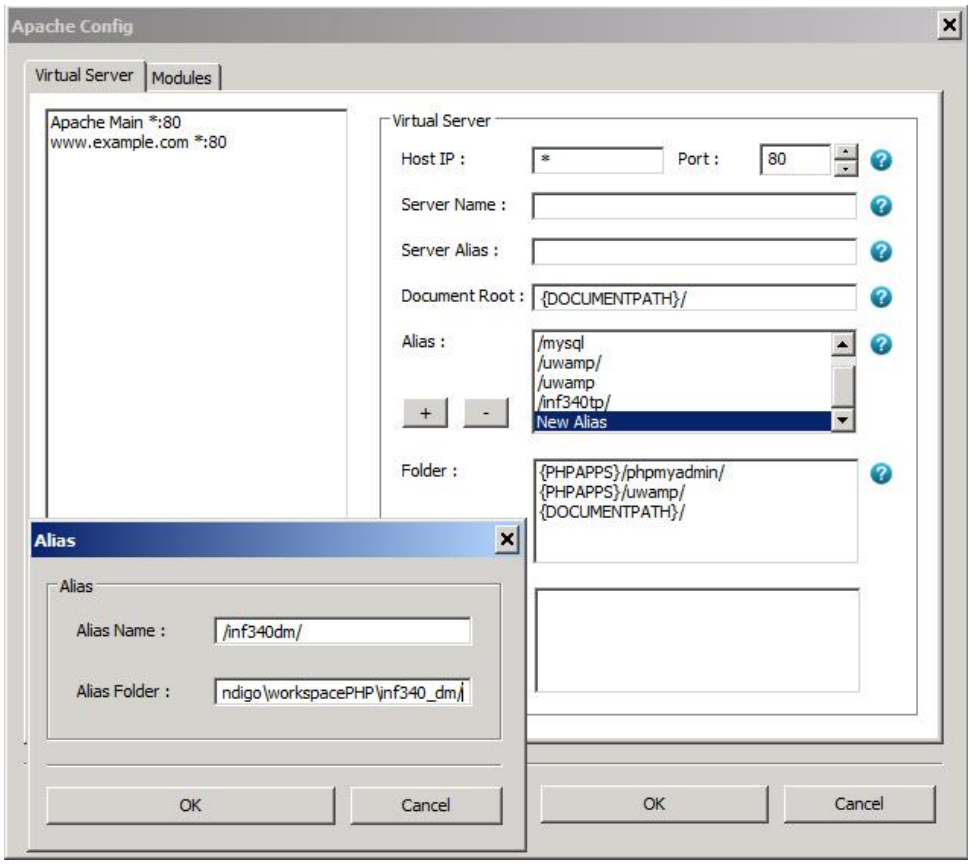
Figure A.11. Import du projet



3.2. Création de l'allias

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

Figure A.12. Création de l'allias



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 crée 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")
     * @JoinColumns({
     *   @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\repositeries;

class UtilisateurRepository extends \Doctrine\ORM\EntityRepository {


}

?>
```

```
<?php


namespace models\repositeries;

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\repositeries\UtilisateurRepository")
```

```
@Entity(repositoryClass="\models\repositeries\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 regénérer notre base de donnée 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 constructeur 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\repositeries\ImageRepository")
 * @HasLifecycleCallbacks
 */
```

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

}
```