

# Prérequis pour SYMFONY et exercices préparatoires

## Table des matières

1.	Objectifs de la séquence .....	2
1.1.	Vers Symfony.....	2
1.2.	Elaboration d'un CRUD.....	2
2.	Choix de la thématique .....	3
2.1.	Pourquoi une classe User et une classe UserManager ?.....	3
2.2.	Vers votre projet « e-boutique ».....	3
3.	Convention de nommage .....	4
4.	La Classe User (User.class.php) .....	5
4.1.	La classe User sera l'image la table users de la base de données.....	5
4.2.	Travail à faire .....	6
5.	La classe UserManager (UserManager.class.php).....	7
5.1.	Travail à faire .....	7
6.	Hydratation.....	8
6.1.	Travail à faire .....	8
7.	CRUD par l'Entity Manager.....	9
7.1.	Travail à faire .....	9
7.1.1.	Créer d'une base de données MySQL et de la table « users » .....	9
7.1.2.	Ajouter la Connexion à la base de données, .....	10
7.1.3.	Compléter la classe UserManager ci-dessus de toutes ses méthodes.....	11
8.	Formulaire .....	12
8.1.	Travail à faire .....	15

# 1. OBJECTIFS DE LA SEQUENCE

## 1.1. VERS SYMFONY

---

Le Framework Symfony distingue les entités et les manageurs d'entités :

➔ Ce TP est un travail préparatoire à l'étude de Symfony

## 1.2. ELABORATION D'UN CRUD

---

CRUD pour Create Read Update Delete est l'acte élémentaire de l'informatique de gestion de la partie Back d'une application Web.

➔ Maîtrise du CRUD

## **2. CHOIX DE LA THEMATIQUE**

### **2.1. POURQUOI UNE CLASSE USER ET UNE CLASSE USERMANAGER ?**

➔ Principe de Séparation des responsabilités

### **2.2. VERS VOTRE PROJET « E-BOUTIQUE »**

(c'est pour plus tard)

➔ Quel que soit votre e-boutique, l'entité User sera toujours nécessaire.

### 3. CONVENTION DE NOMMAGE

Chacun peut avoir ses préférences mais ici, puisque nous allons partager nos réalisations, nous en fixons une pour tous :

- ➔ camelCase et
- ➔ anglais.

Ce qui est important, c'est d'avoir un choix définitif partagé de bout en bout du projet.

## 4. LA CLASSE USER (USER.CLASS.PHP)

### 4.1. LA CLASSE USER SERA L'IMAGE LA TABLE USERS DE LA BASE DE DONNEES.

➔ Noter que la table est au pluriel et la classe au singulier

#	Nom	Type
<input type="checkbox"/> 1	<b>id</b> 	int(11)
<input type="checkbox"/> 2	<b>email</b>	varchar(45)
<input type="checkbox"/> 3	<b>password</b>	varchar(300)
<input checked="" type="checkbox"/> 4	<b>firstName</b>	varchar(45)
<input checked="" type="checkbox"/> 5	<b>lastName</b>	varchar(45)
<input type="checkbox"/> 6	<b>address</b>	varchar(45)
<input type="checkbox"/> 7	<b>postalCode</b>	varchar(45)
<input type="checkbox"/> 8	<b>city</b>	varchar(45)
<input type="checkbox"/> 9	<b>admin</b>	tinyint(1)

User
id: numeric email: varchar password: varchar firstName: varchar lastName: varchar address: varchar postalCode: numeric city: varchar
setId(Sid) getId( ) setEmail (in Semail) getEmail ( ) setFirstNane (\$firstName) getFirstName ( ) setLastName ( \$lastName ) getLastName ( ) setAddress (\$address) getAddress ( ) : setPostalCode (\$postalCode) getPostalCode ( ) setCity (\$city) getCity()

## 4.2. TRAVAIL À FAIRE

---

- ➔ Ajouter ses getters (méthode chargée de renvoyer la valeur d'un attribut)
- ➔ Ajouter ses setters (méthode chargée d'assigner une valeur à un attribut en vérifiant son intégrité).
- ➔ On ajoutera aussi des contrôles à certains setters :
  - On vérifiera ici que firstName et lastName sont bien des chaînes de caractères

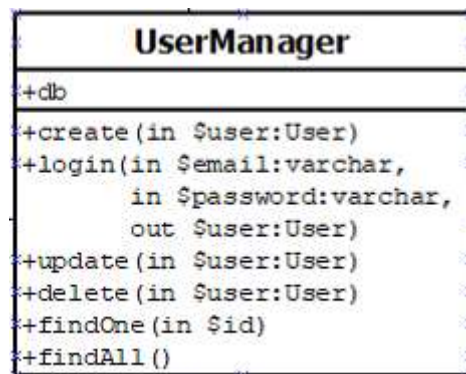
```
<?php
class User {
    private $id;
    private $email;
    private $password;
    private $firstName;
    private $lastName;
    private $address;
    private $postalCode;
    private $city;

    public final function setId($id1) {
        $this->id=$id1;
    }
}
```

- ➔ On placera la classe User complétée de ses getters et setters dans un dossier Model
- ➔ A la racine du site, on créera un fichier index.php
- ➔ Dans ce fichier index.php, on rapatriera la classe User (include ou require)
- ➔ ON pourra tester les méthodes de la classe User

## 5. LA CLASSE UserManager (UserManager.class.php)

La classe UserManager gère toutes les actions avec la Base de Données ou toute autre structure de données (comme les variables de session)



### 5.1. TRAVAIL A FAIRE

A vous de déclarer cette classe UserManager sans détailler ses méthodes, mais avec son constructeur (affectation de l'objet PDO : attribut db)

## 6. HYDRATATION

- ➔ Hydrater un objet, c'est simplement assigner à ses attributs des valeurs (autant que nécessaire pour fonctionner).
- ➔ Ainsi si toutes les attributs ont des valeurs valides, on dit que l'objet est hydraté.

### 6.1. TRAVAIL A FAIRE

Soit un tableau de valeur `$donnees`,

```
<?php
$donnees = array(
    'password'=>'toto',
    'email'=>'Titi@gmail.com',
    'firstName' => 'Patrick',
    'lastName' => 'NOLLET',
    'address' => '4 Place Jussieu',
    'postalCode' => '75252',
    'city' => 'Paris',
    'country' => 'France',
);
```

- ➔ Compléter la méthode hydrate ci-dessous :
- ➔ Utiliser la méthode hydrate pour le constructeur de l'entité User. (Le tableau de données pourra être passé en argument)

```
<?php
class User {
    private $id;
    private $password;
    private $email;
    private $firstName;
    private $lastName;
    private $address;
    private $postalCode;
    private $city;
    private $country;

    public function hydrate(array $donnees) {
        foreach($donnees as $key => $value) {
            // écrire ici le traitement le plus générique possible et le plus
            // sécurisé permettant
            // d'assigner aux attributs les valeurs du tableau de données
        }
    }
}
```



## 7. CRUD PAR L'ENTITY MANAGER


La classe « UserManager » se charge des 4 premières actions constitue le "CRUD"

- Create,
- Read,
- Update,
- Delete

### 7.1. TRAVAIL À FAIRE

#### 7.1.1. CREER D'UNE BASE DE DONNEES MYSQL ET DE LA TABLE « USERS »

Sous PHPMyAdmin, Créer une base de données et sa table « users » avec les champs correspondants cités ci-dessous

#	Nom	Type
<input type="checkbox"/> 1	<b>id</b> 	int(11)
<input type="checkbox"/> 2	<b>email</b>	varchar(45)
<input type="checkbox"/> 3	<b>password</b>	varchar(300)
<input checked="" type="checkbox"/> 4	<b>firstName</b>	varchar(45)
<input checked="" type="checkbox"/> 5	<b>lastName</b>	varchar(45)
<input type="checkbox"/> 6	<b>address</b>	varchar(45)
<input type="checkbox"/> 7	<b>postalCode</b>	varchar(45)
<input type="checkbox"/> 8	<b>city</b>	varchar(45)
<input type="checkbox"/> 9	<b>admin</b>	tinyint(1)

### 7.1.2. AJOUTER LA CONNEXION A LA BASE DE DONNEES,

Placer dans le dossier « Model » la classe « Connection » détaillée ci-dessous.

```
<?php
class Connection {

    private $host;
    private $dbname;
    private $username;
    private $password;
    private $db;

    public function __construct() {

        $this->host = 'localhost';
        $this->dbname = 'dream_seller';
        $this->username = 'root';
        $this->password = 'mysql';
        try
        {
            $this->db = new PDO('mysql:host=' . $this->host . ';dbname='
            . $this->dbname . ';charset=utf8', $this->username, $this->password);
        }
        catch(PDOException $e)
        {
            echo $e->getMessage();
        }
    }

    public function getDb() {
        return $this->db;
    }
}
?>
```

### 7.1.3. COMPLETER LA CLASSE USERMANAGER CI-DESSUS DE TOUTES SES METHODES.

Remplacer toutes les lignes // A compléter par les instructions PHP nécessaires à la méthode déclarée :

- findOne
- update
- delete

On reprendra l'exemple de la méthode findAll proposée ci-dessous et on utilisera la classe « Class Connection » page suivante.

```
<?php
class UserManager {
    private $db;
    public function __construct( $db1 ) {
        $this->db = $db1;
    }
    public function login(User $user) {
        // A compléter
        return $req->fetch();
    }
    public function create(User $user) {
        $req = $this->db->prepare(
            'INSERT INTO users ( lastName, firstName, email, address, postalCode, city,
password, admin )
            VALUES ( :lastName, :firstName, :email, :address, :cp, :city, :password, 0 )'
        );
        $req->execute(
            array(
                'lastName' => $user->getLastName(),
                'firstName' => $user->getFirstName(),
                'email' => $user->getEmail(),
                'address' => $user->getAddress(),
                'cp' => $user->getPostalCode(),
                'city' => $user->getCity(),
                'password' => $user->getPassword()
            )
        );
    }
    public function findAll() {
        $req = $this->db->prepare(
            'SELECT *
            FROM users'
        );
        $req->execute();
        return $req->fetchAll();
    }
    public final function findOne($id) {
        // A compléter
    }
    public final function update(User $user) {
        // A compléter
    }
    public final function delete(User $user) {
        // A compléter
    }
}
```

## 8. FORMULAIRE

Voici un formulaire, il vous est demandé de pouvoir créer un utilisateur à l'aide de ce formulaire et des classes User, UserManager, Connection

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Manager Entité</title>
  <meta name="viewport" content="width=device-width">
  <link href="style.css" rel="stylesheet" type="text/css">
  <!-- Font -->
  <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro"
rel="stylesheet">
  <link href="https://fonts.googleapis.com/css?family=Glegoo"
rel="stylesheet">
</head>
<body>
<header>
  <h1>Création/Modification d'un User</h1>
</header>
<hr/>
<section id="main-section">
  <form action="index.php" method="POST">
    <label>Mail :</label><br/>
    <input type="email" name="email"placeholder="Mail.."/><br>
    <label>Mot de passe :</label><br/>
    <input type="password" name="password"placeholder="Mot de
passe.."/><br>
    <label>Nom :</label><br/>
    <input type="text" name="lastName"placeholder="Nom.."/><br>
    <label>Prénom :</label><br/>
    <input type="text" name="firstName"placeholder="Prénom.."/><br>
    <label>Adresse :</label><br/>
    <input type="text" name="address"placeholder="Adresse.."/><br>
    <label>Code Postal :</label><br/>
    <input type="text" name="postalCode"placeholder="Code
Postal.."/><br>
    <label>Ville :</label><br/>
    <input type="text" name="city"placeholder="Ville.."/><br>
    <p>
      <input type="submit" class="submit-btn" value="Créer/Valider">
    </p>
  </form>
</section>
</body>
</html>
```

# Création/Modification d'un User

---

Mail :

Mot de passe :

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

Créer/Valider

Avec sa feuille de style :

```
/* Minimal properties */
.bold {
    font-weight: bold;
}

/*General*/
body {
    background-color: #f1f1f1;
    min-height: 100%;
    margin: 0;
    font-family: 'Source Sans Pro', sans-serif;
    color: #000;
    text-align: center;
}
hr {
    border: none;
    border-top: 1px solid rgba(0,0,0,0.3);
    width: 80%;
    margin: 20px 10%;
}

/* Title */
h1 {
    font-family: 'Glegoo', serif;
    font-size: 2em;
}

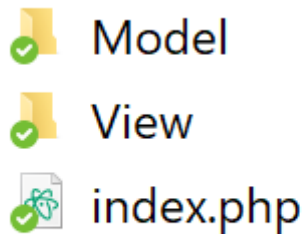
/* Form */
form {
    width: 70%;
    margin: 0 15%;
    text-align: center;
}

/* Buttons */
.submit-btn {
}
```

## 8.1. TRAVAIL À FAIRE

---

Le formulaire (code HTML et CSS dans les pages suivantes), va permettre de tester UserManager et ses méthodes CRUD afin de créer, modifier un user de la base de données.



- Recopier le formulaire et placer le dans un dossier « View »
- Reprendre votre fichier index.php à la racine du site :
  - Rappel1 : ce fichier inclut (include ou require) les classes User, UserManager, Connection
  - Rappel2 : construit une nouvelle instance de la classe Connection
  - Inclut (inclure ou require) le formulaire de création
  - Traite le formulaire de création :
    - Hydrate un nouveau User à partir des données du formulaire
    - Insère ce nouveau User dans la base (UserManager->insert())

Pour rendre votre TP,

- faites un export de votre base de données pour la joindre à vos fichiers
- Ajouter un readme.txt où vous préciserez pour chacun des points demandés (4.2 5.1 6.1 7.1 8.1) ce qui est :
  - OK
  - Fait mais bugé
  - Non Fait
- Faites un zip du tout et déposez le sur la plateforme MOODLE