TD PHP MVC

Étape 10 : Un Repository et des Entities

Nous avons la partie C (controller) de MVC et la partie V (view) avec les fichiers template. Il nous manque la partie M (model). Cette partie se décompose en 2 sous-parties :
Les Entities qui sont des classes correspondant aux tables de la base de données.
Les Repositories qui sont des classes permettant d'interagir avec la base de données (CRUD) et de créer des instances des classes de type Entity.

Nous allons commencer par créer les 2 classes de type Entity correspondant à nos 2 tables en base de données.

Nous créons dans le dossier Entity la classe Category (fichier : category.entity.php) et la classe Product (fichier : product.entity.php) qui sont pour l'instant 2 classes vides.

> */entity/category.entity.php*

entity > category.entity.php > Category

```php
<?php

class Category{

}

```

> */entity/product.entity.php*

entity > product.entity.php > Product

```php
<?php

class Product{

}

```

Nous n'allons créer qu'une seule classe de type Repository (pour siplifier), nous l'appelons MainRepository (fichier : main.repository.php)

> */repository/main.repository.php*

repository > main.repository.php > MainRepository

```php
<?php

class MainRepository{

}

```

Le rôle de cette classe : faire toutes les actions en base de données, peu importe la table et donc l'Entity correspondante en PHP.

Nous définissons donc 2 attributs dans la classe MainRepository qui vont servir à stocker la table et l'entity correspondante et nous initialisons la table dans le constructeur avec un paramètre, l'entity se déduit du nom de la table.

> */repository/main.repository.php*

```php
class MainRepository{

    function __construct($table){
        $this->table = $table;
        $this->entity = ucfirst($this->table);
    }

}
```

Nous allons ensuite extraire de nos controller le code permettant la connexion pour le mettre dans une fonction connect(), fonction private puisqu'elle ne sera utilisé que dans la classe MainRepository. Nous créons auparavant un attribut db pour stocker la connexion et nous l'initialisons à null dans le constructeur.

> */repository/main.repository.php*

```php
class MainRepository{

    function __construct($table){
        $this->table = $table;
        $this->entity = ucfirst($this->table);
        $this->db = null;
    }

    private function connect(){

    }

}
```

PHP *public, protected, private* →

https://www.w3schools.com/php/php_oop_access_modifiers.asp
https://www.php.net/manual/en/language.oop5.visibility.php

Nous allons ensuite récupérer le code que nous avions écrit dans les controller pour la connexion à la base de données et nous l'adaptons pour le mettre dans la fonction connect().

> */repository/main.repository.php*

```php
private function connect(){
    if ($this->db === null) {
        //Connexion à la DB
        $host = "localhost";
        $port = "3306";
        $dbName = "tdphpmvc_db";
        $dsn = "mysql:host=$host;port=$port;dbname=$dbName";
        $user = "root";
        $pass = "";
        $db = null;
        try {
            $db = new PDO(
                $dsn,
                $user,
                $pass,
                array(
                    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                    PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
                )
            );
        } catch (PDOException $e) {
            die("Erreur de connexion à la base de données : $e->getMessage()");
        }
        $this->db = $db;
    }
    return $this->db;
}
```

Nous pouvons supprimer des controller le code servant à la connexion.

> *home.controller.php*

```php
function index(){

    //requête sur la table category
    $sql = "SELECT * FROM category";
    $categories = $db->query($sql)->fetchAll(PDO::FETCH_OBJ);

    $this->entities = ['categories' => $categories];

    $this->render();
}
```

> *product.controller.php*

```php
function index(){

    //requête sur la table product
    $sql = "SELECT * FROM product";
    $products = $db->query($sql)->fetchAll(PDO::FETCH_OBJ);

    $this->entities = ['products' => $products];

    $this->render();
}
```

Nous allons créer maintenant une fonction getAll() dans la classe MainRepository. Cette fonction nous permettra de récupérer toutes les lignes d'une table. Nous pouvons nous inspirer du code déjà écrit dans les controller et le généraliser.

> */repository/main.repository.php*

```php
function getAll(){
    $sql = "SELECT * FROM $this->table";
    $resp = $this->connect()->query($sql);
    return $resp->fetchAll(PDO::FETCH_CLASS, $this->entity);
}
```

Nous avons modifier les paramètres du fetchAll, pour récupérer des instances correspondantes à la classe $this->entity (Product, Category) au lieu d'objet de type StdClass.

Nous allons maintenant modifier nos controller pour utiliser cette fonction. Ne pas oublier re require_once les fichiers contenant la classe MainRepository et les classes de type Entity nécessaires (Product, Category)

(Voir page suivante ...)

> *home.controller.php*

controller > home.controller.php > HomeController > index

```php
<?php

require_once 'base.controller.php';
require_once $_SERVER['DOCUMENT_ROOT'] . '/repository/main.repository.php';
require_once $_SERVER['DOCUMENT_ROOT'] . '/entity/category.entity.php';

class HomeController extends BaseController{

    function index(){

        $repository = new MainRepository("category");
        $categories = $repository->getAll();

        $this->entities = ['categories' => $categories];

        $this->render();
    }

}
```

> *product.controller.php*

controller > product.controller.php > ProductController > index

```php
<?php

require_once 'base.controller.php';
require_once $_SERVER['DOCUMENT_ROOT'] . '/repository/main.repository.php';
require_once $_SERVER['DOCUMENT_ROOT'] . '/entity/product.entity.php';

class ProductController extends BaseController{

    function index(){

        $repository = new MainRepository("product");
        $products = $repository->getAll();

        $this->entities = ['products' => $products];

        $this->render();
    }

}
```

N’oublier pas de re-tester vos 2 routes /home et /product dans Chrome.