

## TD PHP MVC

### Étape 9 : Un peu d'héritage et de re-factorisation du code

Nous allons reprendre notre ProductController pour afficher la liste des produits en base de données. Le code dans le constructeur est exactement le même que pour HomeController.

> *product.controller.php*

```
class ProductController{

    function __construct($params){

        $this->action = $params[0] ?? 'index';
        $this->id = $params[1] ?? null;

        if(!method_exists(get_called_class(), $this->action)){
            $controllerName = get_called_class();
            die("Action \"{$this->action}\" of the Controller \"{$controllerName}\" not exists");
        }

        /*structure du chemin des templates par défaut
        |   template/nameOfTheController/nameOfTheAction.view.php
        */
        $templatesFolder = lcfirst(str_replace("Controller", "", get_called_class()));
        $this->template = "template/{$templatesFolder}/{$this->action}.view.php";

        $this->{$this->action}();

    }
}
```

Pour la fonction index (action par défaut du controller) le code est quasiment identique également. La connexion à la base de données ne change pas. La récupération du template dans le buffer non plus. Seule la requête sera différente, puisque cette fois ci nous allons récupérer des données dans la table product et non plus category.

(Voir page suivante ...)

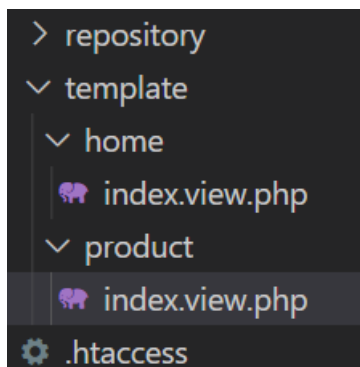
> product.controller.php

```
function index(){
    //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()");
    }


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

    ob_start();
    include_once $this->template;
    $this->content = ob_get_clean();
}
```

Il ne nous reste plus qu'à créer le template correspondant, à savoir un fichier index.view.php dans un sous-dossier product de template



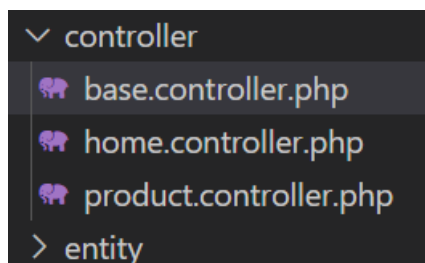
> template/product/index.view.php

template > product >  index.view.php

```
1 <h1>
2     Liste des produits
3 </h1>
4 <ul>
5     <?php
6         foreach ($products as $product) {
7     ?>
8         <li>
9             <?= $product->name ?>
10        </li>
11    <?php
12        }
13    ?>
14 </ul>
```

Cette fois ci, la variable dans laquelle nous avons stocké les données dans le controller s'appelle \$products, attention donc a bien utiliser \$products dans le template.

Pour éviter de recopier n fois le même code, nous allons utiliser l'héritage entre classes. Nous créons une classe BaseController qui sera héritée par tous les autres controller.



Puis nous recopions le constructeur de HomeController ou ProductController (le code est le même dans les 2 controller)

(Voir page suivante ...)

> base.controller.php

```
controller > 🐘 base.controller.php > 📄 BaseController > 📦 __construct
1  <?php
2
3  class BaseController
4  {
5      public function __construct($params)
6      {
7          $this->action = $params[0] ?? 'index';
8          $this->id = $params[1] ?? null;
9
10         if (!method_exists(get_called_class(), $this->action)) {
11             $controllerName = get_called_class();
12             die("Action \"\$this->action\" of the Controller \"\$controllerName\" not exists");
13         }
14
15         /*structure du chemin des templates par défaut
16         |   template/nameOfTheController/nameOfTheAction.view.php
17         */
18         $templatesFolder = lcfirst(str_replace("Controller", "", get_called_class()));
19         $this->template = "template/$templatesFolder/$this->action.view.php";
20
21         $this->{$this->action}();
22     }
23 }
24
25 ?>
```

Nous allons faire hériter tous nos controller (HomeController et ProductController) de la classe BaseController et nous supprimerons leur constructeur afin que ce soit le constructeur de leur classe mère (BaseController) qui soit appelé lors d'une instantiation.

> home.controller.php

```
controller > 🐘 home.controller.php > 📄 HomeController
1  <?php
2
3  require_once 'base.controller.php';
4
5  class HomeController extends BaseController{
6
7      function index(){
8
```

> product.controller.php

```
controller > 🐘 product.controller.php > 📄 ProductController > 📦 index
1  <?php
2
3  require_once 'base.controller.php';
4
5  class ProductController extends BaseController{
6
7      function index(){
8
```

Pensez à faire le `require_once` du fichier contenant la classe `BaseController` (ligne 3)

Vous pouvez tester les routes `/home` et `/product` dans Chrome pour vérifier qu'elles fonctionnent encore.

**PHP extends** → [https://www.w3schools.com/php/keyword\\_extends.asp](https://www.w3schools.com/php/keyword_extends.asp)  
→ <https://www.php.net/manual/en/language.oop5.inheritance.php>

Nous allons également mutualiser la partie du code qui récupère le template dans un buffer et le mutualiser en le mettant dans une fonction `render()` de la classe `BaseController`

Nous supprimons le code suivant des fonctions `index` de nos controller `HomeController` et `ProductController` pour le mettre dans une fonction `render()` de `BaseController`

```
        ob_start();  
        ... include_once $this->template;  
        ... $this->content = ob_get_clean();  
    }
```

> *base.controller.php*

```
23  
24     public function render(){  
25         ob_start();  
26         include_once $this->template;  
27         $this->content = ob_get_clean();  
28     }  
29 }
```

Puis nous appelons cette fonction à la fin de la fonction `index()` de tous nos controller enfants

> *home.controller.php*

```
31     //requête sur la table category  
32     $sql = "SELECT * FROM category";  
33     $categories = $db->query($sql)->fetchAll(PDO::FETCH_OBJ);  
34  
35     $this->render();  
36 }
```

> *product.controller.php*

```
31     //requête sur la table product  
32     $sql = "SELECT * FROM product";  
33     $products = $db->query($sql)->fetchAll(PDO::FETCH_OBJ);  
34  
35     $this->render();  
36 }
```

Il nous reste à stocker les données dans un attribut des classes controller (\$this->entities)  
Il contiendra toutes les données dans un tableau associatif où les clés seront les noms des variables à utiliser dans le template et dont les valeurs seront les données elles-même.

> *home.controller.php*

```
//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*

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

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

$this->render();
```

Enfin, nous modifions la fonction render() de BaseController pour qu'elle récupère ces « entities » sous forme de variable et les insert dans le template.

> *base.controller.php*

```
public function render(){
    foreach($this->entities as $k => $v){
        ... ${$k} = $v;
        ... }

    ob_start();
    include_once $this->template;
    $this->content = ob_get_clean();
}
```

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