

## **TD PHP MVC**

### **Étape 11 : Autoload et page « 404 Not Found ».**

Pour éviter de faire des `require_once` pour chaque classe utilisée dans un fichier, il existe une fonction `spl_autoload_register` permettant d'exécuter une fonction, que nous devons définir, à chaque fois que PHP rencontre une classe lors de l'exécution du code.

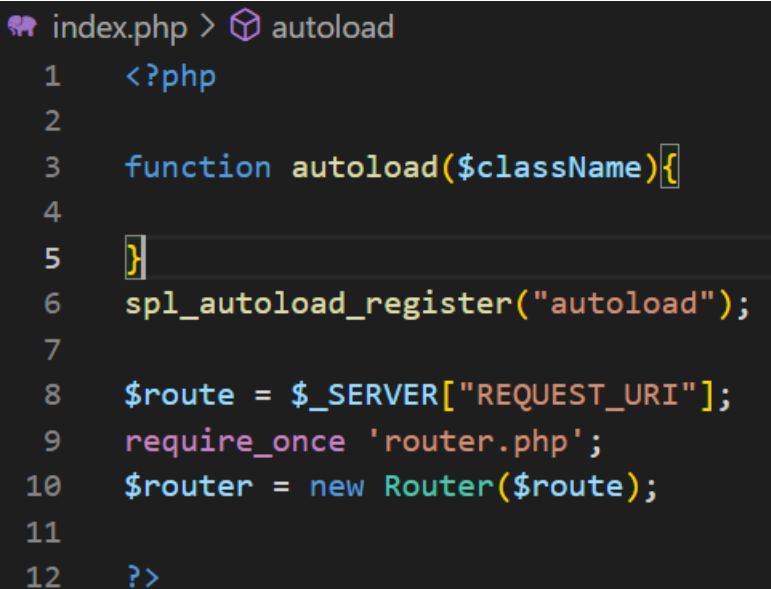
#### **PHP `spl_autoload_register` →**

<https://www.php.net/manual/en/function.spl-autoload-register.php>

<http://www.lephpfacile.com/manuel-php/function.spl-autoload-register.php>

Nous allons donc, au début de notre page d'entrée dans l'application : `index.php`, créer notre fonction `autoload`, puis utiliser `spl_autoload_register` pour la déclencher automatiquement à chaque fois qu'une classe sera rencontrée dans l'exécution du code.

> `index.php`



```
index.php > autoload
1  <?php
2
3  function autoload($className){
4
5  }
6  spl_autoload_register("autoload");
7
8  $route = $_SERVER["REQUEST_URI"];
9  require_once 'router.php';
10 $router = new Router($route);
11
12 ?>
```

le paramètre `$className` correspondra au nom de la classe rencontrée lors de l'exécution du code. A partir du nom de la classe nous pouvons déduire le chemin du fichier qu'il faut inclure avec un `require_once`

(Voir page suivante ...)

> index.php

```
index.php > ...
1  <?php
2
3  function autoload($className){
4      if (strpos($className, "Controller")) {
5          $fileName = lcfirst(str_replace("Controller", "", $className));
6          require_once "controller/$fileName.controller.php";
7      }
8      else if (strpos($className, "Repository")){
9          $fileName = lcfirst(str_replace("Repository", "", $className));
10         require_once "repository/$fileName.repository.php";
11     }
12     else {
13         $fileName = lcfirst($className);
14         require_once "entity/$fileName.entity.php";
15     }
16 }
17 spl_autoload_register("autoload");
18
```

PHP strpos → [https://www.w3schools.com/php/func\\_string\\_strpos.asp](https://www.w3schools.com/php/func_string_strpos.asp)

PHP str\_replace → [https://www.w3schools.com/php/func\\_string\\_str\\_replace.asp](https://www.w3schools.com/php/func_string_str_replace.asp)

PHP lcfirst → [https://www.w3schools.com/php/func\\_string\\_lcfirst.asp](https://www.w3schools.com/php/func_string_lcfirst.asp)

On peut maintenant enlever les require\_once dans nos classes de type controller :-). Nous pouvons également retirer le require\_once sur le controller dans le router.

D'une manière générale, nous n'auront plus besoin de faire des require\_once pour toutes les classes de type Repository, Controller ou Entity, car nous avons traité ces cas dans notre autoload.

Encadré en rouge ci-dessous les require\_once à supprimer. Souligné en vert le chemin du fichier (et éventuellement la fonction) dans lequel faire la modification

```
controller > home.controller.php > ...
1  <?php
2
3  require_once 'base.controller.php';
4  require_once $_SERVER['DOCUMENT_ROOT'] . '/repository/main.repository.php';
5  require_once $_SERVER['DOCUMENT_ROOT'] . '/entity/category.entity.php';
6
7  class HomeController extends BaseController{
8
```

```

controller > product.controller.php > ...
1  <?php
2
3  require_once 'base.controller.php';
4  require_once $_SERVER['DOCUMENT_ROOT'] . '/../repository/main.repository.php';
5  require_once $_SERVER['DOCUMENT_ROOT'] . '/../entity/product.entity.php';
6
7  class ProductController extends BaseController{
8

```

```

router.php > Router > __construct
22
23     if(!file_exists($controllerFilePath)){
24         die("File for the Controller \"{$controllerName}\" not found");
25     }
26
27     require_once $controllerFilePath;
28
29     /*structure des noms de classes correspondants aux controllers
30     NameController
31     */

```

Voilà pour l'autoload, passons maintenant à notre page d'erreur 404 Not Found. Actuellement si le chemin saisi dans l'url ne correspond pas à un controller ou à une action, nous arrêtons le script avec un die. Nous allons plutôt rediriger l'utilisateur vers une page 404. Il faut donc que nous créions une route /error404 et donc un controller Error404Controller ainsi qu'un template pour l'action par défaut (index) : template/error404/index.view.php

Le controller :

```

controller > error404.controller.php > Error404Controller
1  <?php
2
3  class Error404Controller extends BaseController{
4
5      function index(){
6          $this->render();
7      }
8
9  }

```

Le template :

```
template > error404 > 🐘 index.view.php
1 <h1>Erreur 404 : Cette page n'existe pas !</h1>
2
```

Pas besoin d'entités pour cette page, il va falloir que nous initialisons l'attribut entities de BaseController avec un tableau vide, pour que la fonction render (de cette même classe) ne déclenche pas d'erreur lors du foreach sur entities

Nous remplaçons également le die par une redirection avec header dans la classe BaseController (Encadré en vert ci dessous, ce qui à changé dans BaseController)

```
controller > 🐘 base.controller.php > 🗑️ BaseController > 🏠 __construct
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            header("Location: /error404");
12            die;
13        }
14
15        $this->entities = [];
16
17        /*structure du chemin des templates par défaut
18        |   template/nameOfTheController/nameOfTheAction.view.php
19        */
```

Idem dans la classe Router :

```
🐘 router.php > 🗑️ Router > 🏠 __construct
20
21     $controllerFilePath = "controller/$controllerName.controller.php";
22
23     if(!file_exists($controllerFilePath)){
24         header("Location: /error404");
25         die;
26     }
27
28     /*structure des noms de classes correspondants aux controllers
29     |   NameController
30     */
```

Testez dans le navigateur avec une route existante (/home ou product) et avec plusieurs routes n'existant pas (/home/bad ou /notExists)