

# PHP Blog Project

---

## 5. MVC : un routeur et des contrôleurs

Nous allons commencer à structurer notre projet en MVC

Nous renommons notre fichier index.php en home.php et créons un nouveau fichier index.php. Ce nouveau fichier index sera le point d'entrée de notre application.

Toutes les requêtes doivent être dirigées vers ce fichier qui les transmettra au routeur chargé d'appeler le bon contrôleur et d'exécuter la bonne méthode en fonction de la route présente dans l'url.

Pour cela nous ajoutons un fichier .htaccess dans notre répertoire app.

```
app > .htaccess
1 Options -MultiViews
2 RewriteEngine On
3 RewriteCond %{REQUEST_FILENAME} !-f
4 RewriteCond %{REQUEST_FILENAME} !-d
5 RewriteRule ^ index.php [QSA,L]
```

Dans un premier temps, nous allons écrire ce code dans index.php

```
app > index.php > ...
1 <?php
2 $route = filter_var(trim($_SERVER["REQUEST_URI"], '/'), FILTER_SANITIZE_URL);
3 $routeParts = explode('/', $route);
4 $controllerName = ucfirst(array_shift($routeParts));
5 if(empty($controllerName)){
6     $controllerName = "Home";
7 }
8 echo "Controller Name : $controllerName<br/>";
9 $actionName = array_shift($routeParts) ?? "index";
10 echo "Action Name: $actionName<br/>";
11 $params = $routeParts;
12 echo "Params : ";
13 var_dump($params);
```

Ligne 2 : Nous récupérons la route (ce qui suit le nom de domaine) dans la variable prédéfinie `$_SERVER`, nous supprimons les "/" au début et à la fin (à l'aide de la fonction `trim`) et nettoyons avec `filter_var` (sécurité)

Ligne 3 à 13 : Nous découpons la route pour en récupérer les différentes parties et les affichons.

doc :

<https://www.php.net/manual/en/function.trim>

<https://www.php.net/manual/en/function.explode.php>

<https://www.php.net/manual/fr/function.ucfirst.php>

<https://www.php.net/manual/fr/function.array-shift>

<https://www.php.net/manual/fr/function.empty>

La première partie de la route correspondra au contrôleur à appeler, si la route est vide nous utiliserons le contrôleur de la page d'accueil.

La deuxième partie correspond à l'action (méthode) du contrôleur qui doit être utilisée.

Si cette deuxième partie est manquante, nous appellerons une méthode par défaut (index).

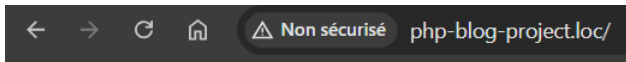
Le reste de la route correspondra à d'éventuels paramètres (ex : l'id d'un article)

La structure d'une route aura donc cette forme

[http://php-blog-project.loc/called\\_controller/called\\_method/param1/param2/etc](http://php-blog-project.loc/called_controller/called_method/param1/param2/etc)

Exemples :

Pour les route <http://php-blog-project.loc/> ou <http://php-blog-project.loc/home>



Controller Name : Home  
Action Name: index  
Params : array(0) { }

Pour la route <http://php-blog-project.loc/articles>

Controller Name : Articles  
Action Name: index  
Params : array(0) { }

Pour la route <http://php-blog-project.loc/articles/details>

Controller Name : Articles  
Action Name: details  
Params : array(0) { }

Pour la route <http://php-blog-project.loc/articles/details/123>

Controller Name : Articles  
Action Name: details  
Params : array(1) { [0]=> string(3) "123" }

Pour la route <http://php-blog-project.loc/articles/details/123/abc>

Controller Name : Articles  
Action Name: details  
Params : array(2) { [0]=> string(3) "123" [1]=> string(3) "abc" }

Nous allons définir toutes le routes de notre application

pour la page d'accueil (liste des derniers articles) -> / ou **/home**

pour la page liste des séries -> **/series**

pour la page liste des technologies -> **/techs**

pour la page liste des articles d'une série donnée -> **/series/articles/:id**

pour la page liste des articles d'une technologie donnée -> **/techs/articles/:id**

pour la page détail d'un article -> **/articles/details/:id**

## La classe Router

Nous ajoutons un répertoire controllers dans lequel nous allons maintenant créer notre classe Router (également appelé Front Controller)

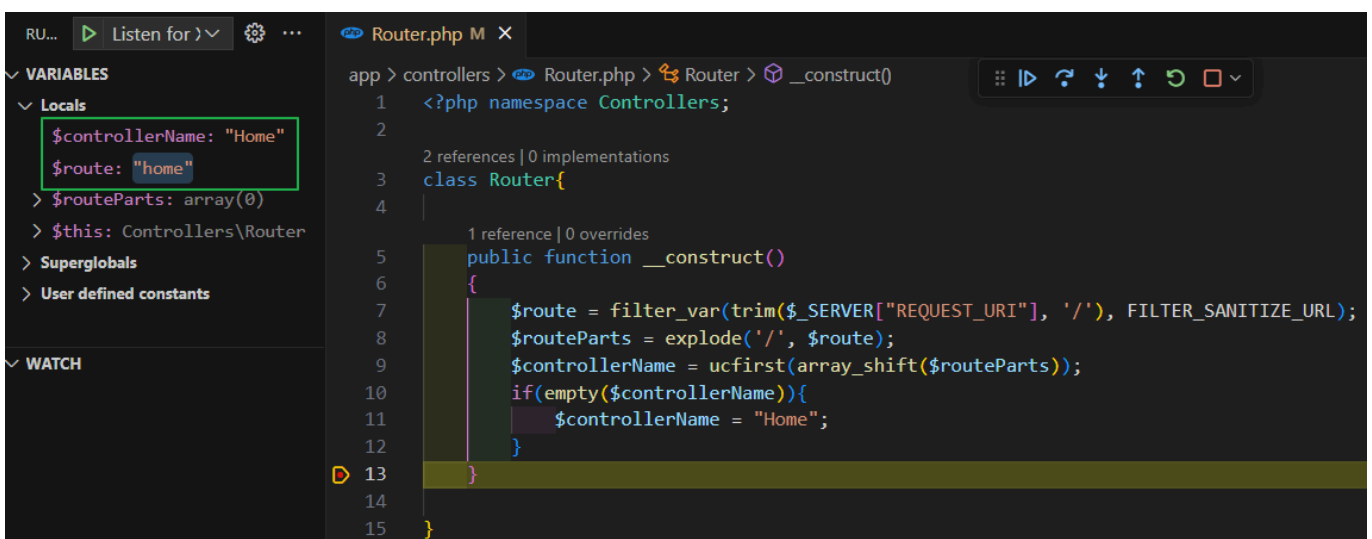
Dans le constructeur nous commençons par récupérer le nom du contrôleur

```
app > controllers > Router.php > ...
1  <?php namespace Controllers;
2
3  class Router{
4
5      1 reference | 0 overrides
6      public function __construct()
7      {
8          $route = filter_var(trim($_SERVER["REQUEST_URI"], '/'), FILTER_SANITIZE_URL);
9          $routeParts = explode('/', $route);
10         $controllerName = ucfirst(array_shift($routeParts));
11         if(empty($controllerName)){
12             $controllerName = "Home";
13         }
14     }
15 }
```

Puis nous créons une instance de la classe Router dans index.php afin de pouvoir tester notre code.

```
12     echo "Params : ";
13     var_dump($params);
14
15     include_once "controllers/Router.php";
16     $router = new Controllers\Router();
```

En mode debug (avec XDebug) avec un point d'arrêt ligne 13 pour la route [/home](#) nous pouvons visualiser les valeurs des différentes variables



The screenshot shows the XDebug interface with the 'Locals' panel on the left and the code editor on the right. The 'Locals' panel displays the following variables:

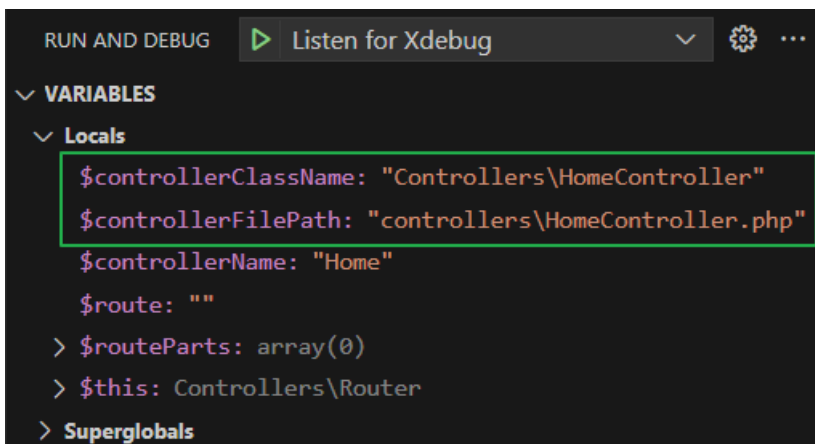
- `$controllerName`: "Home"
- `$route`: "home"
- `$routeParts`: array(0)
- `$this`: Controllers\Router

The code editor shows the `__construct()` method of the `Router` class. The execution is paused at line 13, which is the closing brace of the `__construct()` method. The breadcrumb at the top indicates the current execution context: `app > controllers > Router.php > Router > __construct()`.

Pour utiliser XDebug il faut installer l'extension PHPDebug puis lire les instructions sur la page de l'extension pour la configuration.

A partir du nom du contrôleur, nous allons pouvoir obtenir le nom de la classe (avec son namespace) ainsi que le chemin du fichier contenant la classe.

```
5 public function __construct()
6 {
7     $route = filter_var(trim($_SERVER["REQUEST_URI"], '/'), FILTER_SANITIZE_URL);
8     $routeParts = explode('/', $route);
9     $controllerName = ucfirst(array_shift($routeParts));
10    if(empty($controllerName)){
11        $controllerName = "Home";
12    }
13    $controllerClassName = "Controllers\\".$controllerName."Controller";
14    $controllerFilePath = lcfirst($controllerClassName).".php";
15    if(!file_exists($controllerFilePath)){
16        header('HTTP/1.0 404 Not Found');
17        die();
18    }
19 }
```



Pour l'instant, notre fichier et notre classe HomeController n'existe pas, le script va donc s'arrêter sur la ligne 17.

doc :

<https://www.php.net/manual/en/language.namespaces.nsconstants>

<https://www.php.net/manual/en/function.lcfirst>

<https://www.php.net/manual/en/function.file-exists>

## La classe HomeController

Nous créons notre classe HomeController dans le répertoire app/controllers, fichier HomeController.php

Dans un premier temps nous récupérons le nom de l'action à exécuter à partir des éléments de la route passés en paramètre

```
app > controllers > HomeController.php > HomeController
1  <?php namespace Controllers;
2
   0 references | 0 implementations
3  class HomeController
4  {
   1 reference
5      public $actionName;
   0 references | 0 overrides
6      public function __construct($routeParts)
7      {
8          $this->actionName = array_shift($routeParts) ?? 'index';
9      }
10 }
```

Nous pouvons maintenant importer le fichier du contrôleur dans le routeur et créer une instance de la classe HomeController avec les éléments restant de la route en paramètre (lignes 19 et 20)

```
15     if(!file_exists($controllerFilePath)){
16         header('HTTP/1.0 404 Not Found');
17         die();
18     }
19     include_once $controllerFilePath;
20     $controllerInstance = new $controllerClassName($routeParts);
21 }
22
```

Dans l'exemple que nous avons pris (route [/home](#)) aucune action n'est définie, ce sera donc la méthode par défaut (index) qui doit être exécutée.

```
6      public function __construct($routeParts)
7      {
8          $this->actionName = array_shift($routeParts) ?? 'index';
9      }
10 }
```

```
▼ VARIABLES
  ▼ Locals
    > $routeParts: array(0)
    ▼ $this: Controllers\HomeController
      |   actionName: "index"
    > Superglobals
```

Nous ajoutons une vérification dans le constructeur de la classe HomeController pour s'assurer que la méthode existe dans la classe.

```
6     public function __construct($routeParts)
7     {
8         $this->actionName = array_shift($routeParts) ?? 'index';
9         if (!method_exists(get_called_class(), $this->actionName)) {
10             header('HTTP/1.0 404 Not Found');
11             die();
12         }
13     }
14 }
```

doc :

<https://www.php.net/manual/en/function.method-exists>

<https://www.php.net/manual/en/function.get-called-class>

Puis nous appelons la méthode dans le routeur.

```
18     }
19     include_once $controllerFilePath;
20     $controllerInstance = new $controllerClassName($routeParts);
21     $controllerInstance->{$controllerInstance->actionName}();
22 }
23 }
```

Pour l'instant, la méthode index n'existe pas dans le contrôleur, le script s'arrête ligne 11 de notre contrôleur

```
6     public function __construct($routeParts)
7     {
8         $this->actionName = array_shift($routeParts) ?? 'index';
9         if (!method_exists(get_called_class(), $this->actionName)) {
10             header('HTTP/1.0 404 Not Found');
11             die();
12         }
13     }
14 }
```

Nous allons donc créer une méthode index qui (pour l'instant) affichera un message dans la page

-> voir page suivante

```

app > controllers > HomeController.php > HomeController
1  <?php namespace Controllers;
2
   0 references | 0 implementations
3  class HomeController
4  {
   2 references
5      public $actionName;
   0 references | 0 overrides
6      public function __construct($routeParts)
7      {
8          $this->actionName = array_shift($routeParts) ?? 'index';
9          if (!method_exists(get_called_class(), $this->actionName)) {
10             header('HTTP/1.0 404 Not Found');
11             die();
12         }
13     }
   0 references | 0 overrides
14
15     public function index(){
16         echo "<br/>Executing ".get_called_class()." -> ".__FUNCTION__."()";
17     }
18 }
19

```

doc :

<https://www.php.net/manual/en/language.constants.magic.php>

## Résultat

← → ↺ 🏠 ⚠ Non sécurisé php-blog-project.loc/home

Controller Name : Home

Action Name: index

Params : array(0) { }

Executing Controllers\HomeController -> index()

Si nous testons avec une autre route, rien ne s'affiche puisque seul le controller HomeController existe avec une méthode index.

← → ↺ 🏠 ⚠ Non sécurisé php-blog-project.loc/autre/route

Controller Name : Autre

Action Name: route

Params : array(0) { }

Il va donc falloir créer tous les contrôleurs et leurs méthodes pour les routes que nous avons définies dans notre application :

- SeriesControlleur -> index()
- SeriesControlleur->articles() avec l'id en paramètre
- TechsControlleur -> index()
- TechsControlleur ->articles() avec l'id en paramètre
- ArticlesControlleur->details() avec l'id en paramètre

## La classe ArticlesController

Pour ArticlesController->details() nous avons besoin de récupérer un id de type entier, nous stockons donc le premier élément restants de la route dans \$params et vérifions (dans la méthode details, lignes 18 à 22) que l'id de l'url est bien présent et qu'il est de type entier .

```
app > controllers > ArticlesController.php > ArticlesController
1  <?php namespace Controllers;
2
   0 references | 0 implementations
3  class ArticlesController
4  {
   2 references
5      public $actionName;
   2 references
6      private $params;
   0 references | 0 overrides
7      public function __construct($routeParts)
8      {
9          $this->actionName = array_shift($routeParts) ?? 'index';
10         if (!method_exists(get_called_class(), $this->actionName)) {
11             header('HTTP/1.0 404 Not Found');
12             die();
13         }
14         $this->params = $routeParts;
15     }
16
   0 references | 0 overrides
17     public function details(){
18         $id = (int)$this->params[0];
19         if($id < 1) {
20             header('HTTP/1.0 404 Not Found');
21             die();
22         }
23         echo "<br/>Executing ".get_called_class()." -> ".__FUNCTION__."() with id=".$id;
24     }
25 }
```

Si aucun id n'est présent dans l'url ou que l'id n'est pas un entier, le script s'arrête ligne 21.

doc :

<https://www.php.net/manual/en/language.types.type-juggling.php#language.types.typecasting>

## Résultat

```
← → ↺ 🏠 ⚠ Non sécurisé php-blog-project.loc/articles/details/1

Controller Name : Articles
Action Name: details
Params : array(1) { [0]=> string(1) "1" }
Executing Controllers\ArticlesController -> details() with id=1
```

A vous de coder les classes SeriesController et TechsController avec leurs méthodes index et articles

git : [https://github.com/DWWM-23526/PHP\\_BLOG\\_PROJECT/tree/Step05](https://github.com/DWWM-23526/PHP_BLOG_PROJECT/tree/Step05)