

# PHP Blog Project

## 9. MVC : Correction des bugs et refactoring

Avant d'aller plus loin, nous allons mettre un peu d'ordre dans notre code. Les fichiers home.php et articles.php se trouvant à la racine de app peuvent être supprimés. Nous supprimons également le code commenté (si ça n'est pas déjà fait) dans le fichier index.php et dans les contrôleurs.

### Correction des bugs

Nous allons maintenant nous attaquer à la correction des bugs. A la fin du dernier TP nous avons détecté un bug sur la route articles/details/id

Si l'id passé dans la route n'existe pas en DB ou s'il est omis (dans l'url) nous avons des messages d'erreur qui s'affichent sur notre page.

Le problème vient donc de la méthode details() dans la classe ArticlesController. Nous plaçons des points d'arrêt et allons debugger le code.

1er cas de bug : l'id n'est pas présent dans la route

```
7 public function details(){
8     $id = (int)$this->params[0];
9     if($id < 1) {
10         header('HTTP/1.0 404 Not Found');
11         die();
12     }
13     $articleRepository = new ArticleRepository();
14     $article = $articleRepository->getOneById($id);
15     $attributes = [
16         'article' => $article,
17         'pageTitle' => "MyBlog - Article : ".$article->title,
18     ];
19     $this->render($attributes);
20 }
21 }
```

▼ VARIABLES

▼ Locals

- \$article = uninitialized
- \$articleRepository = uninitialized
- \$attributes = uninitialized
- \$id = uninitialized

▼ \$this = Controllers\ArticlesController

- actionName = "details"

▼ params = array(0)

> Superglobals

En arrêtant l'exécution sur la première ligne nous constatons que \$this->params est un tableau vide dont nous essayons de convertir l'élément 0 en nombre entier. Si nous laissons l'exécution du code se poursuivre, un message est affiché sur notre page.

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

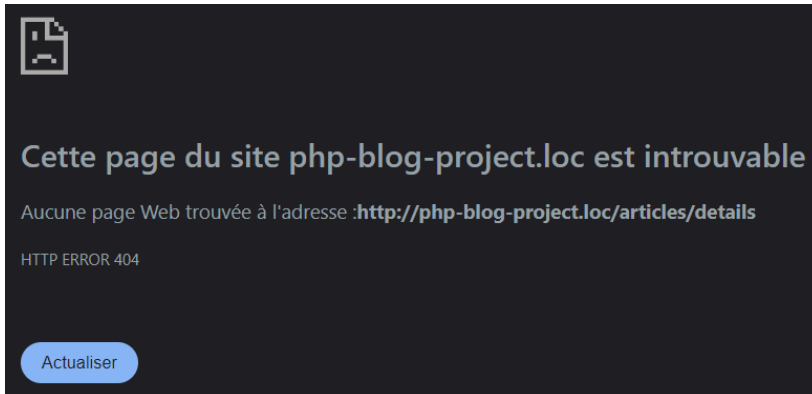
Notice: Undefined offset: 0 in C:\wamp64\www\php\_blog\_project\app\controllers\ArticlesController.php on line 8

Pour corriger le problème, nous vérifions que `$this->params` n'est pas vide avant de faire la conversion. Dans un premier temps nous renvoyons une erreur 404 et stoppons l'exécution du script si tel n'est pas le cas.

Si nous testons à nouveau sans id dans la route

```
7      public function details(){
8          if(empty($this->params)){
9              header('HTTP/1.0 404 Not Found');
10             die();
11         }
12         $id = (int)$this->params[0];
```

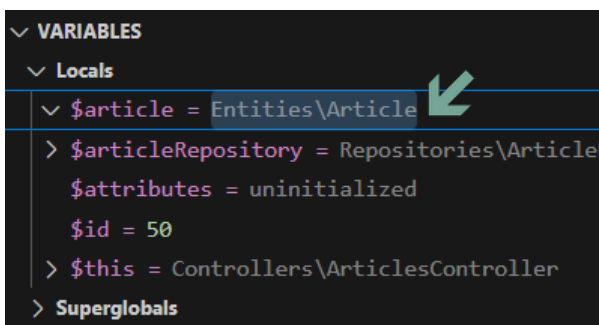
Résultat



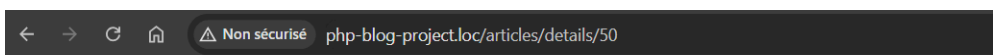
Nous améliorerons cela plus tard avec une redirection vers une page d'erreur par exemple.

2eme cas de bug : l'id est présent dans la route mais ne correspond à aucun article en DB  
Toujours dans notre méthode `details()`, l'appel à la méthode `getOneById()` de la classe `ArticleRepository` renvoie bien un objet de type `Article` mais complètement vide !

```
16      }
17      $articleRepository = new ArticleRepository();
18      $article = $articleRepository->getOneById($id);
19      $attributes = [
20          'article' => $article,
21          'pageTitle' => "MyBlog - Article : ".$article->title,
22      ];
23      $this->render($attributes);
24  }
```



En l'état, une fois le code complètement exécuté, nous obtenons un message comme celui-ci



Warning: Invalid argument supplied for foreach() in C:\wamp64\www\php\_blog\_project\app\entities\Article.php on line 16

Fatal error: Uncaught Error: Typed property Entities\Article::\$title must not be accessed before initialization in C:\wamp64\www\php\_blog\_project\app\controllers\Router.php(27): Controllers\ArticlesController->details() #1 C:\wamp64\www\php\_blog\_project\app\controllers\ArticlesController.php on line 21

En effet, ligne 21 dans notre méthode `details()`, nous essayons d'accéder à la propriété `title` de l'article qui ne contient aucune propriété.

Nous allons donc corriger cela dans la méthode `getOneById()` de la classe `ArticleRepository`. Nous commençons par stocker la réponse du `fetch` dans une variable intermédiaire `$assocArray`

Si cette réponse vaut `false`, ce qui signifie qu'aucune ligne en DB ne correspond à la requête SQL. Dans ce cas nous renvoyons la valeur `null`. Sinon nous renvoyons une instance de la classe `Article` avec les valeurs récupérées dans `$assocArray`.

```
18 public function getOneById($id){
19     $queryResponse = $this->preparedQuery("SELECT * FROM article WHERE id_article = ?", [$id]);
20     $assocArray = $queryResponse->statement->fetch(PDO::FETCH_ASSOC);
21     if(!$assocArray){
22         return null;
23     }
24     $article = new Article($assocArray);
25     return $article;
26 }
```

**VARIABLES**

**Locals**

- `$article` = uninitialized
- `$assocArray` = false
- `$id` = 50
- `$queryResponse` = stdClass
- `$this` = Repositories\ArticleRepository

**Superglobals**

Il nous reste à ajouter une condition dans la méthode `details()` de la classe `ArticlesController` pour vérifier si l'article obtenu par la méthode `getOneById()` est `null` (ou pas). Dans le cas où nous ne récupérons aucun article de la DB (`$article == null`), nous renvoyons une erreur 404.

```
17 $articleRepository = new ArticleRepository();
18 $article = $articleRepository->getOneById($id);
19 if($article == null){
20     header('HTTP/1.0 404 Not Found');
21     die();
22 }
23 $attributes = [
24     'article' => $article,
25     'pageTitle' => "MyBlog - Article : ".$article->title,
26 ];
27 $this->render($attributes);
28 }
```

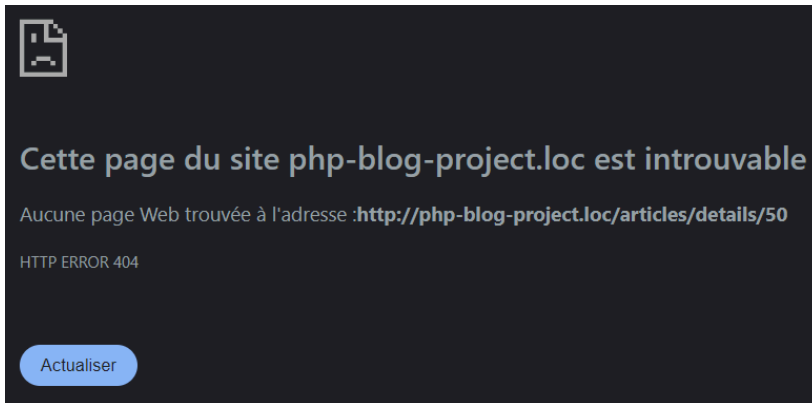
**VARIABLES**

**Locals**

- `$article` = null
- `$articleRepository` = Repositories\ArticleRepository
- `$attributes` = uninitialized
- `$id` = 50
- `$this` = Controllers\ArticlesController

**Superglobals**

## Résultat



Nous devons également corriger les méthodes `getOneById()` dans les autres repositories (Serie et Tech) ainsi que les méthodes `articles()` dans `SeriesController` et `TechsController` pour lesquels un id correct est nécessaire dans l'URL et dont le comportement est le même en cas d'absence d'id ou d'id inexistant en DB.

Autre possibilité, mutualiser le code commun ou très ressemblant entre les différentes classes et en profiter pour corriger les bugs.

## Refactoring du code

Commençons par les repositories.

Dans les méthodes `getAll()` et `getOneById()`, seuls les noms de table et les noms des entités varient. Nous allons donc commencer par essayer de récupérer ces valeurs dans la classe `BaseRepository` en y ajoutant les 3 méthodes ci-dessous.

```
app > repositories > BaseRepository.php > BaseRepository
5  class BaseRepository
39  2 references
   private function getBaseClassName(){
40      $baseClassName = str_replace("Repositories\\", "", get_called_class());
41      return str_replace("Repository", "", $baseClassName);
42  }
43  2 references
   private function getTableName(){
44      return lcfirst($this->getBaseClassName());
45  }
46
47  2 references
   private function getEntityClassName(){
48      return "Entities\\" . $this->getBaseClassName();
49  }
```

`getBaseClassName()` permet de récupérer le nom de base de la classe à partir du Repository appelé, (ex : Article, Serie ou Tech)

`getTableName()` permet de récupérer le nom de la table à partir de `getBaseClassName()` en passant la première lettre en minuscule.

Enfin, `getEntityClassName()` permet de récupérer le nom de l'entité en ajoutant le namespace devant le nom de base de la classe.

Puis nous allons créer une méthode `getAll()` générique qui utilise ces méthodes pour savoir dans quelle table exécuter la requête en DB et quelle classe d'entity produire.

```
50 |
    3 references | 0 overrides
51 | public function getAll(){
52 |     $queryResponse = $this->preparedQuery("SELECT * FROM ".$this->getTableName());
53 |     $entities = $queryResponse->statement->fetchAll(PDO::FETCH_CLASS, $this->getEntityClassName());
54 |     return $entities;
55 | }
```

Il reste à supprimer (commenter) les méthodes `getAll()` dans nos 3 repositories déjà créés (Article, Serie et Tech)

Pour tester nous pouvons temporairement remplacer l'appel à la méthode `getLastPublishedArticles()` dans la classe `HomeController` par la méthode `getAll()`

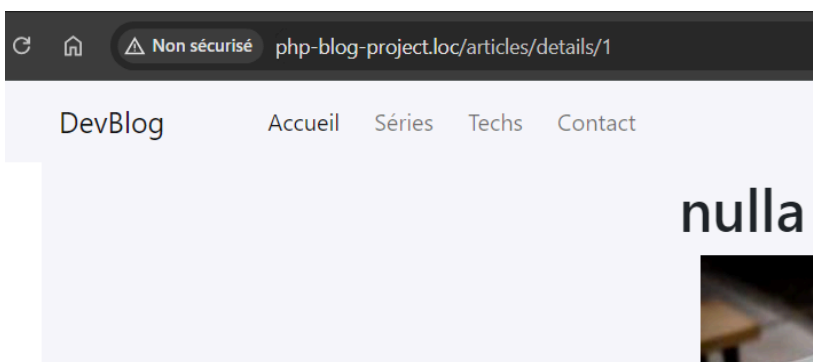
```
4 | class HomeController extends BaseController
5 | {
    0 references | 0 overrides
6 | public function index(){
7 |     $articleRepository = new ArticleRepository();
8 |     // $articles = $articleRepository->getLastPublishedArticles(12);
9 |     $articles = $articleRepository->getAll();
10 |     $attributes = [
```

Nous allons maintenant créer une méthode `getOneById()` générique sur le même principe, toujours dans la classe `BaseRepository`

```
56 |
    3 references | 0 overrides
57 | public function getOneById($id){
58 |     $tableName = $this->getTableName();
59 |     $entityClassName = $this->getEntityClassName();
60 |     $queryResponse = $this->preparedQuery("SELECT * FROM $tableName WHERE id=$tableName = ?", [$id]);
61 |     $assocArray = $queryResponse->statement->fetch(PDO::FETCH_ASSOC);
62 |     if(!$assocArray){
63 |         return null;
64 |     }
65 |     $entity = new $entityClassName($assocArray);
66 |     return $entity;
67 | }
```

Nous pouvons supprimer les méthodes `getOneById()` de nos classes filles de type `Repository` et tester sur la route `articles/details/:id`

Tout semble bien fonctionner.



Dans les classes filles de type Repository, il ne reste plus que les méthodes spécifiques tels que `getLastPublishedArticles()` dans `ArticleRepository`

Il reste encore à corriger les méthodes `articles()` des classes `SeriesController` et `TechsController`. Avant cela, nous allons encore améliorer la vérification de l'id dans la méthode `details()` de la classe `ArticlesController`

Comme vous le constatez, nous retrouvons 3 fois quasiment le même code

```
8 public function details(){
9     if(empty($this->params)){
10         header('HTTP/1.0 404 Not Found');
11         die();
12     }
13     $id = (int)$this->params[0];
14     if($id < 1) {
15         header('HTTP/1.0 404 Not Found');
16         die();
17     }
18     $articleRepository = new ArticleRepository();
19     $article = $articleRepository->getOneById($id);
20     if($article == null){
21         header('HTTP/1.0 404 Not Found');
22         die();
23     }
24     $attributes = [
25         'article' => $article,
26         'pageTitle' => "MyBlog - Article : ".$article->title,
27     ];
28     $this->render($attributes);
29 }
```

La première occurrence peut être supprimée en modifiant légèrement la façon dont nous convertissons le paramètre `$this->params[0]` en nombre entier. Si celui-ci est vide nous le remplaçons par 0 grâce à l'opérateur `??`


```
public function details(){
    // if(empty($this->params)){
    //     header('HTTP/1.0 404 Not Found');
    //     die();
    // }
    // $id = (int)$this->params[0];
    $id = (int)($this->params[0] ?? 0);
    if($id < 1) {
```

Pour éviter les 2 autres répétitions, nous allons créer une classe chargée d'envoyer une réponse HTTP et d'éventuellement arrêter le code.


-> voir page suivante

## La classe HttpResponse

Nous créons une classe HttpResponse dans un namespace Core (répertoire app/core) avec une première méthode qui permet d'envoyer une réponse 404 Not Found.

```
app > core >  HttpResponse.php >  HttpResponse
1  <?php namespace Core;
2
3  3 references | 0 implementations
4  class HttpResponse
5  {
6      2 references | 0 overrides
7      public static function SendNotFound(bool $condition = true){
8          if($condition){
9              header('HTTP/1.1 404 Not Found');
10             die();
11         }
12     }
13 }
```

Nous pouvons ensuite remplacer les 2 dernières répétitions de code dans la méthode details() de la classe ArticlesController

```
0 references | 0 overrides
public function details(){
    // if(empty($this->params)){
    //     header('HTTP/1.0 404 Not Found');
    //     die();
    // }
    // $id = (int)$this->params[0];
    $id = (int)($this->params[0] ?? 0);
    // if($id < 1) {
    //     header('HTTP/1.0 404 Not Found');
    //     die();
    // }
    HttpResponse::SendNotFound($id < 1); 
    $articleRepository = new ArticleRepository();
    $article = $articleRepository->getOneById($id);
    // if($article == null){
    //     header('HTTP/1.0 404 Not Found');
    //     die();
    // }
    HttpResponse::SendNotFound($article == null); 
    $attributes = [
        'article' => $article,
        'pageTitle' => "MyBlog - Article : ".$article->title,
    ];
    $this->render($attributes);
}
```

Nous pouvons également revoir les méthodes articles() des classes SeriesController et TechsController

## Dans SeriesController

```
0 references | 0 overrides
13 public function articles(){
14     $id = (int)($this->params[0] ?? 0);
15     HttpResponse::SendNotFound($id < 1);
16     $serieRepository = new SerieRepository();
17     $serie = $serieRepository->getOneById($id);
18     HttpResponse::SendNotFound($serie == null);
19 }
20 }
```

## Dans TechsController

```
0 references | 0 overrides
13 public function articles(){
14     $id = (int)($this->params[0] ?? 0);
15     HttpResponse::SendNotFound($id < 1);
16     $techRepository = new TechRepository();
17     $tech = $techRepository->getOneById($id);
18     HttpResponse::SendNotFound($tech == null);
19 }
20 }
```

Il existe encore 2 endroits dans le code où nous pouvons utiliser cette méthode pour l'améliorer

Dans le constructeur du routeur lorsque nous vérifions l'existence du contrôleur à utiliser

```
16 $controllerClassName = "Controllers\\".$controllerName."Controller";
17 $controllerFilePath = lcfirst($controllerClassName).".php";
18 // if(!file_exists($controllerFilePath)){
19 //     header('HTTP/1.0 404 Not Found');
20 //     die();
21 // }
22 → HttpResponse::SendNotFound(!file_exists($controllerFilePath));
23 $this->controllerInstance = new $controllerClassName($routeParts);
24
25 }
```

Dans le constructeur de la classe BaseController lorsque nous vérifions l'existence de la méthode à appeler

```
0 references | 0 overrides
8 public function __construct($routeParts)
9 {
10     $this->actionName = array_shift($routeParts) ?? 'index';
11     // if (!method_exists(get_called_class(), $this->actionName)) {
12     //     header('HTTP/1.0 404 Not Found');
13     //     die();
14     // }
15     → $actionNotExists = !method_exists(get_called_class(), $this->actionName);
16     HttpResponse::SendNotFound($actionNotExists);
17     $this->params = $routeParts;
18 }
```

Nous pouvons encore améliorer notre code en créant une méthode pour renvoyer une réponse 200 OK dans notre classe HttpResponse



```

1 reference | 0 overrides
12 public static function SendOK(string $content = null, bool $condition = true){
13     if($condition){
14         header('HTTP/1.1 200 OK');
15         if(isset($content)) echo $content;
16         die();
17     }
18 }
19 }

```

Par rapport à la méthode SendNotFound(), nous devons, en plus du status code (404, 200) afficher le contenu de la vue qui sera stocké dans la variable \$content

Nous allons donc légèrement modifier la méthode render() de notre classe BaseController pour stocker dans une variable la vue à l'aide d'un buffer et utiliser la méthode SendOK pour envoyer le contenu.

```

2 references | 0 overrides
20 protected function render($attributes = [], $viewPath = null){
21     extract($attributes);
22     if(!isset($viewPath)){
23         $controllerName = str_replace("Controller", "", get_called_class());
24         $controllerName = lcfirst(str_replace("s\\", "", $controllerName));
25         $viewPath = "views/pages/$controllerName.$this->actionName.php";
26     }
27     ob_start();
28     require_once $viewPath;
29     $content = ob_get_clean();
30     HttpResponse::SendOK($content);
31 }

```

doc :

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

<https://www.php.net/manual/en/function.ob-start.php>

<https://www.php.net/manual/en/function.ob-get-clean.php>

Il vous reste un peu de travail de refactoring en créant une classe mère **BaseEntity** pour mutualiser le code similaire à toutes vos classes de type Entity

Il reste également un bug à corriger. Notre client s'est rendu compte que la page d'accueil n'affiche pas les 12 derniers articles publiés mais les 12 premiers articles stockés DB.

A vous de jouer !

git :

[https://github.com/DWWM-23526/PHP\\_BLOG\\_PROJECT/tree/Step09](https://github.com/DWWM-23526/PHP_BLOG_PROJECT/tree/Step09)