

# PHP Blog Project

## 7. MVC : les classes Repository et Entity

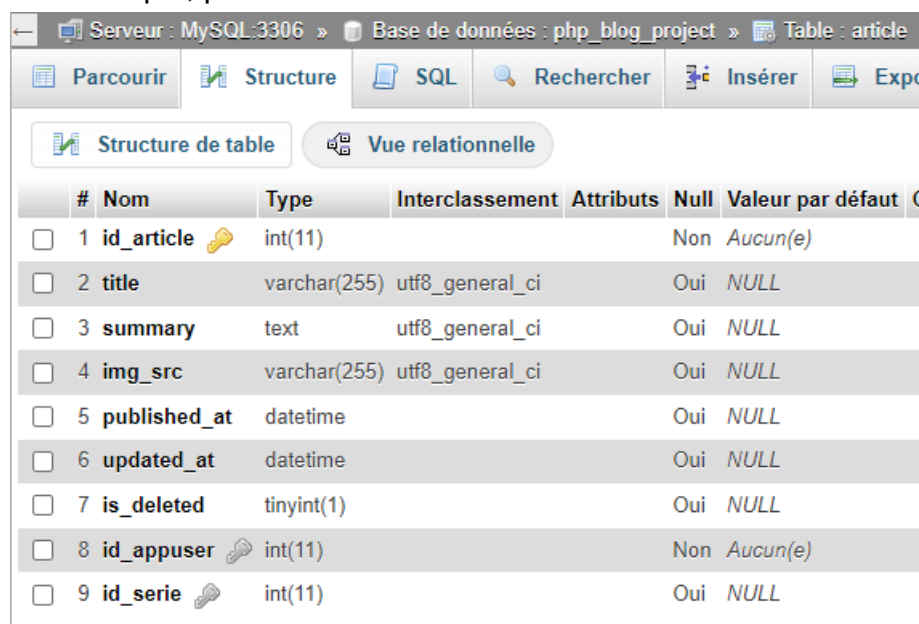
Lors des derniers TP (5 et 6) nous avons vu la partie C (Controller) du MVC, à savoir les contrôleurs et le routeur également appelé “front controller”

Nous allons maintenant aborder la partie M (Model) avec les repositories et les entities.

### La classe Article (Entity)

Une **Entity** (des entities) est une classe représentant une table en base de données.

Par exemple, pour la table article en DB



The screenshot shows the MySQL Table Structure for the 'article' table in the 'php\_blog\_project' database. The table has 9 columns: id\_article (primary key, int(11)), title (varchar(255), utf8\_general\_ci), summary (text, utf8\_general\_ci), img\_src (varchar(255), utf8\_general\_ci), published\_at (datetime), updated\_at (datetime), is\_deleted (tinyint(1)), id\_appuser (foreign key, int(11)), and id\_serie (foreign key, int(11)).

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut
1	id_article	int(11)			Non	Aucun(e)
2	title	varchar(255)	utf8_general_ci		Oui	NULL
3	summary	text	utf8_general_ci		Oui	NULL
4	img_src	varchar(255)	utf8_general_ci		Oui	NULL
5	published_at	datetime			Oui	NULL
6	updated_at	datetime			Oui	NULL
7	is_deleted	tinyint(1)			Oui	NULL
8	id_appuser	int(11)			Non	Aucun(e)
9	id_serie	int(11)			Oui	NULL

Nous créons une classe Article dans le namespace Entities (répertoire entities) avec les champs de la table et un constructeur.

-> voir page suivante

```

app > entities > Article.php > Article
1  <?php namespace Entities;
2
   2 references | 0 implementations
3  class Article
4  {
   0 references
5      public int $id_article;
   0 references
6      public ?string $title;
   0 references
7      public ?string $summary;
   0 references
8      public ?string $img_src;
   0 references
9      public ?string $published_at;
   0 references
10     public ?string $updated_at;
   0 references
11     public ?bool $is_deleted;
   0 references
12     public int $id_appuser;
   0 references
13     public ?int $id_serie;
14
   1 reference | 0 overrides
15     function __construct($fields = []){
16         foreach($fields as $k => $v){
17             if(property_exists($this, $k))
18                 $this->{$k} = $v;
19         }
20     }
21
22 }

```

Le tableau \$fields passé en argument du constructeur sert à initialiser les différentes valeurs, nous vérifions que la classe possède bien un attribut avant de lui affecter sa valeurs.

Les attributs sont typés (int, string, bool). Le point d'interrogation qui précède le type permet de rendre nullable l'attribut en fonction de la nullabilité définie en DB. Dans notre exemple, seuls id\_article et id\_appuser ne sont pas nullable en DB.

doc :

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

## La classe mère BaseRepository

Une **Repository** (des repositories) est une classe responsable des interactions avec la DB. Elle génère lors des requêtes SELECT des instances de type Entity.

Nous commençons par créer une classe mère BaseRepository dans le namespace Repositories (répertoire repositories)

Cette classe sera chargée d'établir la connexion à la DB et d'exécuter les requêtes préparées.

Nous commençons par traiter la connexion

```
app > repositories > BaseRepository.php > BaseRepository
1  <?php namespace Repositories;
2      use PDO;
3      use PDOException;
4
5  4 references | 1 implementation
6  class BaseRepository
7  {
8      3 references
9      private static $connection = null;
10     1 reference
11     private function connect(){
12         if (self::$connection == null) {
13             include_once "../configs/db.config.php";
14             //Connexion à la DB
15             $dsn = "mysql:host=".DB_HOST.";port=".DB_PORT.";dbname=".DB_NAME;
16             $user = DB_USER;
17             $pass = DB_PASSWORD;
18             try {
19                 $connection = new PDO(
20                     $dsn,
21                     $user,
22                     $pass,
23                     array(
24                         PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
25                         PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
26                     )
27                 );
28             } catch (PDOException $e) {
29                 $errorMessage = $e->getMessage();
30                 die("Erreur de connexion à la base de données : $errorMessage");
31             }
32             self::$connection = $connection;
33         }
34         return self::$connection;
35     }
36 }
```

Pour utiliser les classes PDO et PDOException nous les importons (lignes 2 et 3)

La connexion doit être unique, nous la stockons dans une variable static \$connection et n'en créons une nouvelle dans la méthode connect que si elle n'existe pas déjà.

Puis nous ajoutons la méthode `preparedQuery` qui renvoie un objet contenant la requête et le résultat (true ou false) de son exécution.

```
29     self::$connection = $connection;
30 }
31 return self::$connection;
32 }
33 3 references | 0 overrides
34 protected function preparedQuery($sql, $params = []){
35     $statement = $this->connect()->prepare($sql);
36     $result = $statement->execute($params);
37     return (object)['result' => $result, 'statement' => $statement];
38 }
```

doc :

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

<https://www.php.net/manual/en/language.oop5.static.php>

<https://www.php.net/manual/en/language.oop5.paamayim-nekudotayim.php>

<https://www.php.net/manual/en/language.oop5.late-static-bindings.php>

## La classe ArticleRepository

Cette classe sera plus spécialisée, elle hérite de la classe `BaseRepository` et sera chargée des interactions avec la table article en DB.

Elle générera des instances de la classe `Article` lors des requêtes `SELECT`.

Dans un premier temps nous créons une méthode permettant de récupérer tous les articles en DB.

```
app > repositories > ArticleRepository.php > ArticleRepository
1  <?php namespace Repositories;
2      use PDO;
3
4  2 references | 0 implementations
5  class ArticleRepository extends BaseRepository
6  {
7      1 reference | 0 overrides
8      public function getAll(){
9          $queryResponse = $this->preparedQuery("SELECT * FROM article");
10         $articles = $queryResponse->statement->fetchAll(PDO::FETCH_CLASS,"Entities\\Article");
11         return $articles;
12     }
13 }
```

L'utilisation de la méthode `fetchAll` avec pour option `PDO::FETCH_CLASS` et le nom de la classe en 2ème paramètre permet de générer directement des objets de la classe correspondante.

Ne pas oublier l'import de la classe `PDO` (ligne 2)

Nous testons sur notre route "/" ou "/home" (HomeController -> index)

```
app > controllers > HomeController.php > HomeController
1  <?php namespace Controllers;
2  use Repositories\ArticleRepository;
3
4  class HomeController extends BaseController
5  {
6      public function index(){
7          echo "<br/>Executing ".get_called_class()." -> ".__FUNCTION__."()<br/>";
8          $articleRepository = new ArticleRepository();
9          $articles = $articleRepository->getAll();
10         var_dump($articles);
11     }
12 }
```

Dans la méthode index, nous créons une instance de la classe ArticleRepository, puis stockons le résultat de la méthode getAll dans une variable \$articles que nous affichons avec un var\_dump.

Ne pas oublier l'import de la classe ArticleRepository (ligne 2)

Résultat

```
< > ↺ ⌂ ⚠ Non sécurisé php-blog-project.loc/home

Controller Name : Home
Action Name: index
Params : array(0) { }
Executing Controllers\HomeController -> index()
array(48) { [0]=> object(Entities\Article)#7 (9) { ["id_article"]=> int(
congue, risus semper porta volutpat, quam pede lobortis ligula, sit am
["published_at"]=> string(19) "2022-05-09 00:00:00" ["updated_at"]=
```

Sur notre page d'accueil, ce n'est pas tous les articles que nous voulons afficher, mais les 12 derniers articles publiés. Nous ajoutons donc (dans la classe ArticleRepository) une méthode permettant de récupérer en DB les derniers articles publiés.

```
9      return $articles;
10     }
11     1 reference | 0 overrides
12     public function getLastPublishedArticles($qty){
13         $sql = "SELECT * FROM article ORDER BY ? DESC LIMIT $qty;";
14         $queryResponse = $this->preparedQuery($sql, ['published_at']);
15         $articles = $queryResponse->statement->fetchAll(PDO::FETCH_CLASS,"Entities\Article");
16         return $articles;
17     }
18 }
```

Dans le contrôleur, il suffit de remplacer l'appel à la méthode `getAll` par celle que nous venons de créer. Nous passons le nombre d'articles souhaité en paramètre.

```
4 class HomeController extends BaseController
5 {
6     0 references | 0 overrides
7     public function index(){
8         echo "<br/>Executing ".get_called_class()." -> ".__FUNCTION__."()<br/>";
9         $articleRepository = new ArticleRepository();
10        $articles = $articleRepository->getLastPublishedArticles(12);
11        var_dump($articles);
12    }
13 }
```

## Résultat

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

Controller Name : Home
Action Name: index
Params : array(0) { }
Executing Controllers\HomeController -> index()
array(12) { [0]=> object(Entities\Article)#7 (9) { ["id_article"]=> int(
congue, risus semper porta volutpat, quam pede lobortis ligula, sit am
["published_at"]=> string(19) "2022-05-09 00:00:00" ["updated_at"]=
```

Nous allons compléter notre classe `ArticleRepository` avec une méthode permettant de récupérer un article en DB grâce à son id.

```
12 public function getLastPublishedArticles($qty){
13     $sql = "SELECT * FROM article ORDER BY ? DESC LIMIT $qty;";
14     $queryResponse = $this->preparedQuery($sql, ['published_at']);
15     $articles = $queryResponse->statement->fetchAll(PDO::FETCH_CLASS, "Entities\Article");
16     return $articles;
17 }
18 1 reference | 0 overrides
19 public function getOneById($id){
20     $queryResponse = $this->preparedQuery("SELECT * FROM article WHERE id_article = ?", [$id]);
21     $article = new Article($queryResponse->statement->fetch(PDO::FETCH_ASSOC));
22     return $article;
23 }
```

N'oubliez pas d'importer la classe `Article`.

Nous allons tester cette méthode dans la classe ArticlesController méthode details ( ce qui correspond à la route /articles/details/:id )

```
app > controllers > ArticlesController.php > ArticlesController > details()
1  <?php namespace Controllers;
2  use Repositories\ArticleRepository;
3
0 references | 0 implementations
4  class ArticlesController extends BaseController
5  {
6
0 references | 0 overrides
7  public function details(){
8      $id = (int)$this->params[0];
9      if($id < 1) {
10         header('HTTP/1.0 404 Not Found');
11         die();
12     }
13     echo "<br/>Executing ".get_called_class()."-> '._FUNCTION_.'() with id=".$id . "<br/>";
14     $articleRepository = new ArticleRepository();
15     $article = $articleRepository->getOneById($id);
16     var_dump($article);
17 }
18 }
```

N'oubliez pas d'importer la classe ArticleRepository.

## Résultat

← → ↺ 🏠 ⚠ Non sécurisé php-blog-project.loc/articles/details/10

Controller Name : Articles

Action Name: details

Params : array(1) { [0]=> string(2) "10" }

Executing Controllers\ArticlesController -> details() with id=10

object(Entities\Article)#7 (9) { ["id\_article"]=> int(10) ["title"]=> string(19) "eget ac nulla. Sed vel enim sit amet nunc viverra dapibus. Nulla suscipit ligula in lacus. 05 00:00:00" ["updated\_at"]=> string(19) "2023-12-07 00:00:00" ["is\_deleted"]=>

Nous pouvons en profiter pour tester le constructeur de la classe Article que nous n'avions pas encore utilisé jusqu'à maintenant.

```
13  public ?int $id_serie;
14
1 reference | 0 overrides
15  function __construct($fields = []){
16      foreach($fields as $k => $v){
17          if(property_exists($this, $k))
18              $this->{$k} = $v;
19      }
20  }
21
22 }
```

A chaque tour de boucle, vous pouvez voir dans la fenêtre "variables" les différentes valeurs.

```
▼ VARIABLES
  ▼ Locals
    ▼ $fields = array(9)
      id_article = "10"
      title = "eget tincidunt eget"
      summary = "Donec vitae nisi. Nam ultr..."
      img_src = "https://picsum.photos/id/1..."
      published_at = "2022-08-05 00:00:00"
      updated_at = "2023-12-07 00:00:00"
      is_deleted = "1"
      id_appuser = "6"
      id_serie = "8"
      $k = "id_article"
      $v = "10"
    > $this = Entities\Article
  > Superglobals
```

```
▼ VARIABLES
  ▼ Locals
    > $fields = array(9)
      $k = "title"
      $v = "eget tincidunt eget"
    > $this = Entities\Article
  > Superglobals
```

A vous de coder !

Commencez par créer les classes entités Serie et Tech.

Créez ensuite les repositories SerieRepository et TechRepository avec leurs méthodes getAll et getOneById.

Tester les méthodes dans les contrôleurs correspondants. Les méthodes index des contrôleurs affichent la liste des séries ou des techs, les méthodes articles devront dans un premier temps afficher la série ou la tech correspondant à l'id de la route.

git :

[https://github.com/DWWM-23526/PHP\\_BLOG\\_PROJECT/tree/Step07](https://github.com/DWWM-23526/PHP_BLOG_PROJECT/tree/Step07)