PHP Blog Project

10. MVC : BaseEntity et dernières vues (Series et Techs)

La classe BaseEntity

Cette classe reprend le constructeur qui était présent dans les classes de type Entity, avec 2 lignes supplémentaires en début du constructeur afin d'initialiser l'id commun à toutes les tables mais avec un nom variable d'une table à une autre.

app > entities > BaseEntity.php > BaseEntity

```php
<?php namespace Entities;

3 references | 3 implementations
class BaseEntity
{
    0 references | 0 overrides
    function __construct($fields = []){
        $pk = "id_" . lcfirst(str_replace("Entities\\", "", get_called_class()));
        $this->{$pk} = 0;
        foreach($fields as $k => $v){
            if(property_exists($this, $k)){
                $this->{$k} = $v;
            }
        }
    }

}
```

Nos classes de type Entity héritent de BaseEntity. Elles n'ont plus besoin de constructeur, ni de l'attribut id comme le montre la classe Techs ci dessous

app > entities > Tech.php > Tech

```php
<?php namespace Entities;

1 reference | 0 implementations
class Tech extends BaseEntity
{
    // public int $id_tech;
    0 references
    public ?string $label;
    0 references
    public ?string $img_src;
    0 references
    public ?bool $is_deleted;

    // function __construct($fields = []){
    //     foreach($fields as $k => $v){
    //         if(property_exists($this, $k))
    //             $this->{$k} = $v;
    //     }
    // }

}
```

L'id n'étant pas explicitement déclaré comme attribut dans les classes de type Entity, il est nécessaire de combiner PDO::FETCH_CLASS avec PDO::FETCH_PROPS_LATE comme option dans la méthode fetchAll() de la classe BaseRepository. Ceci afin que le constructeur soit exécuté avant l'affectation des valeurs en DB. Un fois le constructeur exécuté, l'id existe (lignes 6 et 7 du constructeur de la classe BaseEntity)

```php
2 references | 0 overrides
public function getAll(){
    $queryResponse = $this->preparedQuery("SELECT * FROM ".$this->getTableName());
    $entities = $queryResponse->statement->fetchAll(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE, $this->getEntityClassName());
    return $entities;
}
```

Nous ajoutons également cette option dans la méthode getLastPublishedArticles() de la classe ArticleRepository et en profitons pour corriger le bug vu en fin de TP 9, en modifiant la requête SQL.

```php
1 reference | 0 overrides
public function getLastPublishedArticles($qty){
    $sql = "SELECT * FROM article ORDER BY published_at DESC LIMIT $qty;";
    $queryResponse = $this->preparedQuery($sql);
    $articles = $queryResponse->statement->fetchAll(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE,"Entities\Article");
    return $articles;
}
```

La vue series.index.php

Nous complétons la méthode index dans le contrôleur

app > controllers > SeriesController.php > SeriesController > index()

```php
class SeriesController extends BaseController
{
    0 references | 0 overrides
    public function index(){
        $serieRepository = new SerieRepository();
        $series = $serieRepository->getAll();
        $attributes = [
            'series' => $series,
            'pageTitle' => "MyBlog - Les séries",
        ];
        $this->render($attributes);
    }
```

Puis codons la vue

app > views > pages > series.index.php

```php
<!DOCTYPE html>
<html lang="fr">
<head>···
</head>
<body>
    <div class="container-lg bg-light">
        <?php include_once "views/partials/navbar.php"; ?>
        <main class="mt-5 pt-3 row">
            <h2>Les séries</h2>
            <?php foreach($series as $serie){ ?>
                <div class="col-12 col-md-6 col-lg-3 d-flex align-items-stretch justify-content-center">
                    <div class="card mb-3" style="width: 18rem;">
                        <img src="<?= $serie->img_src ?>" class="card-img-top"
                            alt="<?= $serie->title ?>">
                        <div class="card-body text-center">
                            <h5 class="card-title"><?= $serie->title ?></h5>
                            <p class="card-text" style="text-align: justify;">
                                <?= $serie->summary?>
                            </p>
                            <a href="series/articles/<?= $serie->id_serie ?>" class="btn btn-primary">Voir</a>
                        </div>
                    </div>
                </div>
            <?php } ?>
        </main>
```

La vue series.articles.php

Nous commençons par écrire la méthode getArticles() dans la classe SerieRepository

app > repositories > SerieRepository.php > SerieRepository

```php
<?php namespace Repositories;
    use PDO;
3 references | 0 implementations
class SerieRepository extends BaseRepository
{
    1 reference | 0 overrides
    public function getArticles($id){
        $sql = "SELECT * FROM article WHERE id_serie = $id";
        $queryResponse = $this->preparedQuery($sql);
        $articles = $queryResponse->statement->fetchAll(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE,"Entities\Article");
        return $articles;
    }
}
```

Cette méthode permet de récupérer les articles pour une série données ($id)
Puis nous complétons la méthode articles du contrôleur.

app > controllers > SeriesController.php > SeriesController > articles()

```php
class SeriesController extends BaseController
    0 references | 0 overrides
    public function articles(){
        $id = (int)($this->params[0] ?? 0);
        HttpResponse::SendNotFound($id < 1);
        $serieRepository = new SerieRepository();
        $serie = $serieRepository->getOneById($id);
        $articles = $serieRepository->getArticles($id);
        HttpResponse::SendNotFound($serie == null);
        $attributes = [
            'serie' => $serie,
            'articles' => $articles,
            'pageTitle' => "MyBlog - Série : ". $serie->title,
        ];
        $this->render($attributes);
    }
}
```

Nous pouvons ensuite écrire coder la vue.

app > views > pages > series.articles.php

```php
<!DOCTYPE html>
<html lang="fr">
<head>···
</head>
<body>
    <div class="container-lg bg-light">
        <?php include_once "views/partials/navbar.php"; ?>
        <main class="mt-5 pt-3 row">
            <h2>Articles de la série "<?= $serie->title?>"</h2>
            <?php foreach($articles as $article){ ?>
                <div class="col-12 col-md-6 col-lg-3 d-flex align-items-stretch justify-content-center">
                    <div class="card mb-3" style="width: 18rem;">
                        <img src="<?= $article->img_src ?>" class="card-img-top"
                            alt="<?= $article->title ?>">
                        <div class="card-body text-center">
                            <h5 class="card-title"><?= $article->title ?></h5>
                            <p class="card-text" style="text-align: justify;">
                                <?= $article->summary?>
                            </p>
                            <a href="../../articles/details/<?= $article->id_article ?>" class="btn btn-primary">Lire</a>
                        </div>
                    </div>
                </div>
            <?php } ?>
        </main>
```

La vue techs.index.php

Même principe que pour la vue series.index.php, méthode index() du contrôleur :

app > controllers > TechsController.php > TechsController > index()

```php
class TechsController extends BaseController
{
    0 references | 0 overrides
    public function index(){
        $techRepository = new TechRepository();
        $techs = $techRepository->getAll();
        $attributes = [
            'techs' => $techs,
            'pageTitle' => "MyBlog - Les techs",
        ];
        $this->render($attributes);
    }

```

Puis la vue.

app > views > pages > techs.index.php

```php
<!DOCTYPE html>
<html lang="fr">
<head>···
</head>
<body>
    <div class="container-lg bg-light">
        <?php include_once "views/partials/navbar.php"; ?>
        <main class="mt-5 pt-3 row">
            <h2>Les techs</h2>
            <?php foreach($techs as $tech){ ?>
                <div class="col-12 col-md-6 col-lg-3 d-flex align-items-stretch justify-content-center">
                    <div class="card mb-3" style="width: 18rem;">
                        <img src="<?= $tech->img_src ?>" class="card-img-top"
                            alt="<?= $tech->label ?>">
                        <div class="card-body text-center">
                            <h5 class="card-title"><?= $tech->label ?></h5>
                            <a href="techs/articles/<?= $tech->id_tech ?>" class="btn btn-primary">Voir</a>
                        </div>
                    </div>
                </div>
            <?php } ?>
        </main>
```

La vue techs.articles.php

Même principe que pour la vue series.articles.php sauf que cette fois ci, la requête SQL qui permet de récupérer les articles doit “passer” par la table de liaison article_tech puisqu’il s’agit d’une relation ManyToMany

app > repositories > TechRepository.php > TechRepository

```php
<?php namespace Repositories;
    use PDO;
3 references | 0 implementations
class TechRepository extends BaseRepository
{
    1 reference | 0 overrides
    public function getArticles($id){
        $sql = "SELECT * FROM article WHERE id_article IN (SELECT id_article FROM article_tech WHERE id_tech = $id)";
        $queryResponse = $this->preparedQuery($sql);
        $articles = $queryResponse->statement->fetchAll(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE,"Entities\Article");
        return $articles;
    }
}
```

Le contrôleur

app > controllers > TechsController.php > TechsController > articles()

```php
class TechsController extends BaseController
    0 references | 0 overrides
    public function articles(){
        $id = (int)($this->params[0] ?? 0);
        HttpResponse::SendNotFound($id < 1);
        $techRepository = new TechRepository();
        $tech = $techRepository->getOneById($id);
        $articles = $techRepository->getArticles($id);
        HttpResponse::SendNotFound($tech == null);
        $attributes = [
            'tech' => $tech,
            'articles' => $articles,
            'pageTitle' => "MyBlog - Tech : ". $tech->label,
        ];
        $this->render($attributes);
    }
}
```

Puis la vue.

app > views > pages > techs.articles.php

```php
<!DOCTYPE html>
<html lang="fr">
<head> ···
</head>
<body>
    <div class="container-lg bg-light">
        <?php include_once "views/partials/navbar.php"; ?>
        <main class="mt-5 pt-3 row">
            <h2>Articles de la tech "<?= $tech->label?>"</h2>
            <?php foreach($articles as $article){ ?>
                <div class="col-12 col-md-6 col-lg-3 d-flex align-items-stretch justify-content-center">
                    <div class="card mb-3" style="width: 18rem;">
                        <img src="<?= $article->img_src ?>" class="card-img-top"
                            alt="<?= $article->title ?>">
                        <div class="card-body text-center">
                            <h5 class="card-title"><?= $article->title ?></h5>
                            <p class="card-text" style="text-align: justify;">
                                <?= $article->summary?>
                            </p>
                            <a href="../../articles/details/<?= $article->id_article ?>" class="btn btn-primary">Lire</a>
                        </div>
                    </div>
                </div>
            <?php } ?>
        </main>
```

Nous pouvons maintenant tester toutes les routes de notre application 🙂

git :
https://github.com/DWWM-23526/PHP_BLOG_PROJECT/tree/Step10