SYMFONY : 4-L'ORM de Symfony : Doctrine

Nous allons maintenant voir comment Symfony gère ses relations avec la base de données.

Pour cela nous allons utiliser l'ORM (objet relationnal mapping), c'est une brique logiciel qui fait la relation entre notre application et une base de données. Le but est de gérer au sein de notre application, via des classes et des objets nos données et ce que l'on fait dans notre application se reflète automatiquement dans la base de données grâce à l'ORM. Le but est de ne presque jamais toucher à notre BDD, nous n'écrirons presque jamais de SQL. On utilisera de simple classe et objets dans notre application et l'ORM se chargera de ce que l'on fait dans notre application se reflète dans la BDD. L'ORM de Symfony s'appelle DOCTRINE.

DOCTRINE n'est pas du tout lié à Symfony, nous pouvons l'utiliser dans d'autre projet PHP.

Grâce à DOCTRINE nous pouvons gérer nos tables, nos champs, les ajouter, les supprimer, les mettre à jour, faire des sélections etc… Voici les outils que nous allons nous servir :

- Entity : des classes qui représentent des tables
- Manager : permet de manipuler les lignes (insertion, mise à jour, suppression)
- Repository : permet de faire des selections (rechercher des articles à une certaine date par exemple)

Tout ça sans avoir à jamais écrire une seule ligne de SQL, c'est DOCTRINE qui se charge de faire tout ça. On ne fait que demander à DOCTRINE ne SQL des choses dont on a besoin.

Dans Symfony on utilise ce que l'on appel des migrations, la philosophie de Symfony est de privilégier les fichiers, les fichiers seront partagés entre différent développeur, sur GIT nous allons télécharger des fichiers et non la BDD, la BDD doit venir des fichiers, les fichiers doivent exprimer à quoi ressemble la BDD.

Une migration c'est tout simplement un script qui nous dit, je veux faire passer la BDD d'un état A à un état B.

- Migration#1 : 1 fichier qui crée 2 tables (tous les fichiers ont un ordre bien précis)
- Migration#2 :
- je modifie les champs d'une table
- j'en crée une autre
- j'en supprime une autre
- Migration#3 :
- Je supprime un champ d'une table
- J'ajoute une relation entre deux tables

Lorsqu'un autre développeur reprendra le dossier, il lui suffira de faire tourner les fichiers de migrations pour qu'il se retrouve avec la même BDD que moi.

Dernier outil à connaitre, ce sont les FIXTURES, c'est un script qui va créer des jeux de fausses données au sein de la BDD, qui est exécutables à souhait et réutilisable par les autres.

En trois lignes de commandes, nous auront une BDD en place, au même niveau avec les mêmes tables et surtout des fausses données pour déjà commencer à travailler et voir votre site internet évolué.

Nous allons démarrer sans BDD. Nous allons tout d'abord ouvrir le fichier .env qui contient les variables d'environnement. Modifier la ligne suivante :

```
DATABASE_URL=mysql://root:@127.0.0.1:3306/demoBlog?serverVersion=5.7
```

Dans le terminal taper la commande suivante pour créer la BDD : (pensez à allumer MYSQL)

```
php bin/console doctrine:database:create
```

Aller dans MYSQL et la BDD est créé !! Ensuite nous allons créer une table via la commande suivante :

```
php bin/console make:entity
```

Entrer ensuite le nom de la classe : **Article**

➔ 2 fichiers crées

created: src/Entity/Article.php (table des articles)

created: src/Repository/ArticleRepository.php (permettra de faire des selections dans la table article)

Nous allons maintenant grâce à des lignes de commandes, créer les différents champs de la table 'Article'

- title → type 'string' (taper enter) → longueur (255 taper enter) → NOT NULL (enter)
- content → type 'text' → NOT NULL (enter)
- image → type 'string' (taper enter) → longueur (255 taper enter) → NOT NULL (enter)
- createdAt → type 'datetime' → NOT NULL (enter)

Observer le fichier Article.php dans le dossier 'Entity', on retrouve les getteurs / setteurs pour chaque champs, avec des propriétés privée de la table 'Article'.

Toutefois, cela ne veut pas dire que la table et les champs ont été créés dans la BDD. Nous allons maintenant créer la migration qui va permettre d'analyser mon code. DOCTRINE va regarder mes entités, elle va voir tout ce qui devrait exister dans la BDD, elle va regarder la BDD vide et les entités et du coup DOCTRINE va dire, il manque telle table avec tel champs donc créons un script SQL pour amener / migrer ce code SQL. Entrer la commande suivante dans le terminal :

```
php bin/console make:migration
```

On peut maintenant observer dans le dossier 'Migrations', un fichier PHP avec le script et les requêtes de création de table SQL, donc un développeur externe n'aura plus qu'à migrer les fichiers pour obtenir la même BDD que moi.

Pour lancer la migration, envoyer la commande suivante dans le terminal :

```
php bin/console doctrine:migrations:migrate
```

cela lance les scripts de migrations afin de mettre à jour la BDD

demande de confirmation répondre : **y** (yes)

Nous allons maintenant créer un fixture, c'est-à-dire que nous allons créer des fausses données pour remplir la table 'Article' et avoir déjà du contenu pour travailler. C'est un script qui va nous permettre de créer un jeu de fausses données dans la BDD.

Il faut tout d'abord installer le composant de fixture, qui n'est pas livré par défaut dans le skeleton, pour cela envoyer la commande suivante dans le terminal :

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

Ensuite :

```
php bin/console make:fixtures
```

Appelé la classe : **ArticleFixtures**

Symfony nous indique qu'il a créé dans le dossier '**DataFixtures'** un fichier **ArticleFixtures.php**. Nous avons un Manager qui a été créé qui nous permettra d'insérer, de modifier ou de supprimer dans la BDD.

Dans le fichier **ArticleFixtures.php**, modifier la méthode load() , ne pas oublier l'appel de la classe Article (use App\Entity\Article)

Extension : php namespace resolver

```
use App\Entity\Article;

class ArticleFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        for($i = 1; $i <= 10; $i++)
        {
            $article = new Article(); // on instancie la class Article() qui s
e trouve dans le dossier App\Entity
            // Nous pouvons maintenant faire appel au setteur pour créer des a
rticles
            $article->setTitle("Titre de l'article n°$i")
                    ->setContent("<p>Contenu de l'article n°$i</p>")
                    ->setImage("http://placehold.it/250x150")
                    -
>setCreatedAt(new \DateTime()); // on instancie la classe DateTime() pour form
ater l'heure

            $manager-
>persist($article); // permet de faire persister l'article dans le temps
        }
```

```
        $manager-
>flush(); // la méthode flush() balance réellement la requête SQL qui mettra e
n place les différentes manipulations que nous avons fait ici
    }
}
```

Pour envoyer les données dans la BDD, entrer la commande suivante dans le terminal :

```
php bin/console doctrine:fixtures:load
```

demande de purge, répondre YES