Doctrine ORM

Doctrine est un framework de Mapping Relationnel-Objet (ORM en anglais). Une tel framework permet de traduire des données représentée dans un modèle orientée objet (utilisé dans les langages de proframmation OO) vers un modèle relationnel (utilisée des les systèmes de gestion de bases de données relationnels). Les ORMs permettent de gagner beaucoup de temps dans une application qui manipule des données, en déchargeant le programmeur d'une grande quantité de code très redondant.

Installation de Doctrine ORM

Installez le package Doctrine via Composer :

```
composer require symfony/orm-pack
composer require --dev symfony/maker-bundle
```

Ensuite, pour configurer les accès à la base de donnée, ouvrez le fichier `.env` à la racine de votre projet et cherchez le paramètre `DATABASE_URL`, et assignez lui la valeur suivante :

```
mysql://db_user:db_password@localhost:3306/db_name?serverVersion=5.7
```

en remplaçant `db_user`, `db_password`, et `db_name` par les valeurs correspondant à votre base.

Créer une Entity

Une Entity est une classe PHP dont les données seront persistées dans dans la base de donnée par Doctrine. Pour en créer, il existe un utilitaire en ligne de commande. Pour y faire appel, utilisez :

```
php bin/console make:entity
Class name of the entity to create or update:
> Product
```

```
New property name (press <return> to stop adding fields): > name
```

```
Field type (enter ? to see all types) [string]: > string
```

```
Field length [255]: > 255
```

```
Can this field be null in the database (nullable) (yes/no) [no]: > no
```

```
New property name (press <return> to stop adding fields): > price
```

```
Field type (enter ? to see all types) [string]: > integer
```

```
Can this field be null in the database (nullable) (yes/no) [no]: > no
```

```
New property name (press <return> to stop adding fields): > (press enter again to finish)
```

L'outil en ligne de commande va ensuite vous demander les champs de votre entité un par un, ainsi que les informations à propos de vos champs (type, taille, etc ...).

Quand vous aurez fini, l'outil va vous créer une classe sous `src/Entity`:

```
namespace App\Entity;
```

```
use App\Repository\ProductRepository; use Doctrine\ORM\Mapping as ORM;
```

```
/**

@ORM\Entity(repositoryClass=ProductRepository::class) / class Product { /*

@ORM\Id()
@ORM\GeneratedValue()
@ORM\Column(type="integer") */ private $id;

/**

@ORM\Column(type="string", length=255) */ private $name;

/**

@ORM\Column(type="integer") */ private $price;

public function getId(): ?int { return $this->id; }

// ... getter and setter methods }
```

Vous pouvez constater que les infos que vous avez données à l'outil ont été converties en annotations, qui vous ensuite permettre à l'ORM dans sérialiser la classe dans une base de donnée relationnelle.

Migrations

Pour pouvoir utiliser l'ORM, il faut d'abord synchroniser le schéma de votre base de donnée avec vos Entity PHP. Pour cela il faut créer puis exécuter une migration.

Créer une migration :

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

Exécuter une migration :

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

Au moment de la migration, Symfony vas exécuter le SQL nécessaire pour mettre à jours son schéma de façon à gérer les Entities.

Persister des données

Pour faire appelle à la couche de persistence dans le controller, il faut faire appel à l'Entity Manager de doctrine. Vous pouvez t accèder dans un controlleur comme ceci :

On peut ensuite créer un objet Entity :

```
$product = new Product();
$product->setName('Vis en acier inoxydable');
$product->setPrice(1999);
$product->setDescription('6mm de diamètre');
```

On peut ensuite persister l'Entity :

```
$entityManager->persist($product);
```

Et enfin commiter la transaction :

```
$entityManager->flush();
```

Requêter des données

Pour récupérer des données via Doctrine, cela se passe avec le Repository de l'Entity, que vous obtenez ainsi :

```
$repository = $this->getDoctrine()->getRepository(Product::class);
```

Vous pouvez ensuite récupérer des Entities par Id :

```
$product = $repository->find($id);
```

Vous pouvez ensuite filtrer sur un ou plusieurs propriétés avec la méthode `findOneBy()` et en lui passant un tableau clé-valeur avec en clé le nom du champs à filtrer et en valeur la valeur sur laquelle filtrer. Exemple :

```
$product = $repository->findOneBy([
    'name' => 'Keyboard',
    'price' => 1999,
]);
```

Mettre à jours des données

Pour mettre à jours une Entity, il suffit de la récupérer avec le Repository, de la modifier puis de commiter les changements avec `$entityManager->flush()` :

```
$entityManager = $this->getDoctrine()->getManager();
$product = $entityManager->getRepository(Product::class)->find($id);
$product->setName('Nouveau nom');
$entityManager->flush();
```

Revision #2
Created 21 November 2024 15:20:46 by Nicolas
Updated 24 November 2024 17:49:04 by Nicolas