La couche modèle et les données : démarrer avec Doctrine

Doctrine est un ORM php (mapping Objet/Relationnel). Les ORM consistent à réaliser la correspondance entre le modèle de données relationnel et le modèle objets de la façon la plus aisée et efficace possible. Un outil d'ORM propose un certain nombre de fonctionnalités parmi lesquelles :

- Assurer le mapping (correspondance) des tables avec les classes, des champs avec les attributs, des relations et des cardinalités
- Proposer une interface qui permette de facilement mettre en oeuvre des actions de type CRUD (Create/Read/Update/Delete)
- Proposer un langage de requêtes indépendant de la base de données cible et assurer une traduction en SQL natif selon la base utilisée
- Supporter différentes formes d'identifiants générés automatiquement par les bases de données (identity, sequence, ...)
- Fournir des fonctionnalités pour améliorer les performances (cache, lazy loading, ...) .

Configuration de la connexion et création de la base de données

Cette étape consiste à indiquer au projet l'adresse de la base de données, son nom, et le login/mot de passe du compte permettant de s'y connecter. Pour cela, vérifier votre fichier de configuration .env (dans c:\wamp64\www\webstudent\ ou à la racine du projet)

```
#DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
```

Exemple :

```
DATABASE_URL="mysql://root:@127.0.0.1:3307/webstudent?serverVersion=mariadb-10.10.2"
```

Cette configuration permet de se connecter à une base de données mariadb nommée webstudent, située sur le poste local. Le compte de connexion est root, sans mot de passe.

Si vous n'avez pas encore créé la base de données sous mariadb ou mysql, la console permet de créer la base de données en exécutant la commande ci-dessous :

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

```
c:\wamp64\www\webstudent>php bin/console doctrine:database:create
Created database `webstudent` for connection named default
```

Créer une entité = classe métier

Dans symfony, les classes métiers se nomment des entités. Elles peuvent être créées avec la console :

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

Le système nous invite alors à saisir l'ensemble des propriétés de l'entité. Pour chaque propriété, nous devons préciser son type, si cette propriété sera `nullable` en bdd ou pas.

Dans l'exemple ci-dessous, nous créons l'entité `Etudiant` avec la propriété `nom` de type `string` de 50 caractères. Vous créerez les propriétés nom, prenom, dateNaiss (type date), ville.

```
c:\wamp64\www\webstudent>php bin/console make:entity

Class name of the entity to create or update (e.g. VictoriousKangaroo):
> Etudiant

created: src/Entity/Etudiant.php
created: src/Repository/EtudiantRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

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

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

Field length [255]:
> 50

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

updated: src/Entity/Etudiant.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>
```

Un fichier nommé Etudiant.php a alors été créé dans le dossier .../src/Entity. Il s'agit d'une classe métier classique avec ses propriétés. Les getters et les setters ont été générés automatiquement. Une propriété supplémentaire id de type int a été ajoutée. Elle est utilisée pour faire le mapping de la clé primaire de la table qui sera un numérique auto-incrémenté.

Créer la table en base de données

Il est alors possible de créer la table correspondant à l'entité.

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

Cette instruction créé à chaque fois un fichier dans le dossier src/migration avec les instructions sql à exécuter.

```
c:\wamp64\www\webstudent>php bin/console make:migration

 Success!

Next: Review the new migration "src/Migrations/Version20181028144417.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
```

Pour exécuter les instructions sql, lancer la commande suivante :

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

```
c:\wamp64\www\webstudent>php bin/console doctrine:migrations:migrate

                    Application Migrations

WARNING! You are about to execute a database migration that could result in schema changes and data loss. Are you sure y
ou wish to continue? (y/n)y
Migrating up to 20181028144417 from 0

  ++ migrating 20181028144417

     -> CREATE TABLE etudiant (id INT AUTO_INCREMENT NOT NULL, nom VARCHAR(50) NOT NULL, prenom VARCHAR(40) NOT NULL, da
te_naiss DATE DEFAULT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci ENGINE = InnoDB

  ++ migrated (0.11s)

  ------------------------

  ++ finished in 0.11s
  ++ 1 migrations executed
  ++ 1 sql queries
```

On voit alors l'instruction sql exécutée. Vérifiez en base l'existence de la table. A chaque propriété de l'entité correspond un champ de la table. Celle ci a pour clé primaire Id.

Persister un objet en base

Il s'agit ici de créer un objet et de rendre ses données persistantes en base de données. Pour cela, nous allons créer une nouvelle url qui exécutera une méthode du contrôleur créant un objet et insérant un enregistrement en base. Le contrôleur redirigera vers une page de consultation de l'enregistrement créé.

Ajout de la nouvelle route

Dans le fichier des routes, ajouter :

```
etudiantAjouter:
    path: /etudiant/ajouter
    controller: App\Controller\EtudiantController::ajouter
```

Modification du contrôleur

Ajout de la méthode ajouterEtudiant dans EtudiantController

```
public function ajouter(ManagerRegistry $doctrine){
    // récupère le manager d'entités
        $entityManager = $doctrine->getManager();

        // instanciation d'un objet Etudiant
        $etudiant = new Etudiant();
        $etudiant->setNom('Potter');
        $etudiant->setPrenom('Harry');
        $etudiant->setDateNaiss(new \DateTime(date('1980-07-31')));
    $etudiant->setVille('Surrey');

        // Indique à Doctrine de persister l'objet
        $entityManager->persist($etudiant);

        // Exécue l'instruction sql permettant de persister lobjet, ici un
INSERT INTO
        $entityManager->flush();

        // renvoie vers la vue de consultation de l'étudiant en passant
l'objet etudiant en paramètre
       return $this->render('etudiant/consulter.html.twig', [
            'etudiant' => $etudiant,]);
    }
```

une erreur apparaitra car la classe métier (l'entité) Etudiant n'est pas connue de EtudiantController. Il faut donc l'importer en ajoutant la ligne ci-dessous au niveau des use (imports) ainsi que le manager d'entités.

```
use App\Entity\Etudiant;
use Doctrine\Persistence\ManagerRegistry;
```

Création de la vue

Dans le dossier templates/etudiant, créer le fichier ci-dessous, nommé consulter.html.twig

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <h5>PAGE DE CONSULTATION D'UN ETUDIANT</h5>
 <p>
  <table >
     <tr><td>Id : </td><td>{{etudiant.id}} </td></tr>
    <tr><td>Nom : </td><td>{{etudiant.nom}}</td>
```

```
    <td  rowspan="8" class="imgEtu">
                    {% set photo = 'img/etudiant/'~ etudiant.id ~'.jpg' %}
                    <span class="zoom"><img src="{{ asset(photo | trans )}}"
title="{{ etudiant.prenom ~ " " ~ etudiant.nom }}" alt="{{ etudiant.prenom ~
" " ~ etudiant.nom }}"/></span></td></tr> </td>
    <tr><td>Prénom : </td><td>{{etudiant.prenom}}</td></tr></tr>
       <tr><td>Date de naissance :
</td><td>{{etudiant.dateNaiss|date("d/m/Y")}}</td></tr>
        <tr><td>rue : </td><td>{{etudiant.numrue}}
{{etudiant.rue}}</td></tr>
         <tr><td>code postal : </td><td>{{etudiant.copos}}</td></tr>
          <tr><td>ville : </td><td>{{etudiant.ville}}</td></tr>
       <tr><td>Surnom : </td><td>{{etudiant.surnom}}</td></tr>
 </table>
    </body>
</html>
```

Note : dans la vue ci-dessus,, des propriétés supplémentaires ont été ajoutées. Ajoutez les dans votre entité étudiant et faites le mapping avec la base de données.

Tests

http://localhost/webstudent/public/etudiant/ajouter

id	nom	prenom	date_naiss	ville
1	Potter	Harry	*NULL*	Surrey

Vérifier la présence de l'enregistrement en base.

Consulter un étudiant

```
    public function consulterEtudiant(ManagerRegistry $doctrine, int $id){

        $etudiant= $doctrine->getRepository(Etudiant::class)->find($id);

        if (!$etudiant) {
            throw $this->createNotFoundException(
            'Aucun etudiant trouvé avec le numéro '.$id
            );
        }

        //return new Response('Etudiant : '.$etudiant->getNom());
        return $this->render('etudiant/consulter.html.twig', [
            'etudiant' => $etudiant,]);
    }
```

La méthode prend en paramètre l'id de l'étudiant permettant la recherche en bdd de l'étudiant selon son id. Il faut donc indiquer dans l'url l'id de l'étudiant. Dans le fichier de routes, il faut donc ajouter le

paramètre :

```
etudiantConsulter:
    path: /etudiant/consulter/{id}
    controller: App\Controller\EtudiantController::consulterEtudiant
```

Lister les étudiants

EtudiantController, méthode permettant de lister les étudiants

```
    public function listerEtudiant(ManagerRegistry $doctrine){

                $repository = $doctrine->getRepository(Etudiant::class);
        $etudiants= $repository->findAll();
        return $this->render('etudiant/lister.html.twig', [
            'pEtudiants' => $etudiants,]);
    }
```

Vue Twig listant les étudiants

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
    </head>
    <body>
        {% block body %}{% endblock %}
        {% block javascripts %}{% endblock %}
        LISTE DES ETUDIANTS</br>
        <table>
        {% for e in pEtudiants %}
        <tr>
            <td><a href="{{ path('etudiantConsulter', { 'id': e.id }) }}">{{
e.nom }}</a></td>
            <td>{{ e.prenom }}</a></td>
            <td>{{ e.dateNaiss|date('d/m/Y') }}</td>
            <td>{{ e.ville}}</td>
    {% else %}
        <tr>
        <td>Aucun etudiant n'a été trouvé.</td>
        </tr>
    {% endfor %}
</table>
    </body>
</html>
```