المعهد العالي للدراسات التكنولوجية بمدنين
Institut Supérieur des Etudes Technologiques de Médenine

Atelier Framework Coté Serveur

TP4

Base de données
(Symfony 6)

Enseignante : K.MECHLOUCH	Classe : L2DSI

1. Introduction

Symfony fournit tous les outils dont vous avez besoin pour utiliser des bases de données dans vos applications grâce à Doctrine, le meilleur ensemble de bibliothèques PHP pour travailler avec des bases de données. Ces outils prennent en charge les bases de données relationnelles telles que MySQL et PostgreSQL, ainsi que les bases de données NoSQL telles que MongoDB.

2. Travail à faire

Le site Web *séminaire* vise à recueillir des commentaires concernant les conférences.
Nous devons stocker les commentaires apportés par les participants à la conférence.
Un commentaire est décrit par une structure de données fixe : un auteur, son e-mail, le texte du commentaire et une photo facultative.
MySQL est le moteur de base de données que nous utiliserons.

1. Installation de Doctrine

Tout d'abord, installez le support de Doctrine via le pack *orm Symfony*, ainsi que le *MakerBundle*, qui vous aidera à générer du code :

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

2. Configuration de la base de données

Les informations de connexion à la base de données (seminaire) sont stockées dans une variable d'environnement appelée ***DATABASE_URL***. Pour le développement, vous pouvez trouver et personnaliser ceci dans le fichier .env :

```
# .env (or override DATABASE_URL in .env.local to avoid committing your changes)
# customize this line!
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7"
# to use mariadb:
DATABASE_URL="mariadb://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=mariadb-10.5.8"
```

```
# to use sqlite:
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/app.db"
# to use postgresql:
#DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
# to use oracle:
# DATABASE_URL="oci8://db_user:db_password@127.0.0.1:1521/db_name"
```

Maintenant que vos paramètres de connexion sont configurés, Doctrine peut créer la base de données db_name pour vous :

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

3. Création d'une classe d'entité

Une conférence peut être décrite avec quelques propriétés :
• La ville où la conférence est organisée ;
• L'année de la conférence ;
• Un drapeau international pour indiquer si la conférence est locale ou internationale.

Vous pouvez utiliser la commande ***make:entity*** pour créer cette classe et tous les champs dont vous avez besoin :

```
symfony console make:entity Conference
```

La commande vous posera quelques questions - répondez-y comme ci-dessous :

- ville, string, 50, no;
- annee, integer, 4, no;
- international, boolean, no.

La classe Conference a été stockée sous *App\Entity*.
La commande a également généré une classe de référentiel Doctrine : *App\Repository\ConferenceRepository*.
Le code généré ressemble à ceci :

```
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
/**
* @ORM\Entity(repositoryClass="App\Repository\ConferenceRepository")
*/
class Conference
{
/**
* @ORM\Id()
* @ORM\GeneratedValue()
* @ORM\Column(type="integer")
*/
private $id;
/**
* @ORM\Column(type="string", length=255)
*/
```

```
private $ville;
// ...
public function getVille(): ?string
{
return $this->ville;
}
public function setVille(string $ville): self
{
$this->ville= $ville;
return $this;
}
// ...
}
```

N.B:
@ORM\Entity(repositoryClass="App\Repository\ConferenceRepository") =
#[ORM\Entity(repositoryClass: ConferenceRepository::class)]

Doctrine ajoute une propriété id pour stocker la clé primaire de la ligne dans la table de la base de données. Cette clé (*@ORM\Id()*) est générée automatiquement (*@ORM\GeneratedValue()*) via une stratégie qui dépend du moteur de la base de données.

Avec la même manière créez maintenant l'entité *Commentaire* :

- auteur, string, 50, no;
- text, text, no;
- email, string, 60, no;
- dateCreat, datetime, no.

4. Lien entre les entités

Les deux entités, *Conference* et *Commentaire*, devraient être liées.
Une conférence peut avoir zéro ou plusieurs commentaires, c'est ce qu'on appelle une relation un à plusieurs.
Utilisez de nouveau la commande *make:entity* pour ajouter cette relation au classe conférence :

```
symfony  console make:entity Conference
Your entity already exists! So let's add some new fields!
New property name (press <return> to stop adding fields):
> comment
Field type (enter ? to see all types) [string]:
> OneToMany
What class should this entity be related to?:
> Commentaire
A new property will also be added to the Comment class...
New field name inside Comment [conference]:
>
Is the Comment.conference property allowed to be null (nullable)? (yes/no)
[yes]:
> no
```

```
Do you want to activate orphanRemoval on your relationship?
A Comment is "orphaned" when it is removed from its related Conference.
e.g. $conference->removeComment($comment)
NOTE: If a Comment may *change* from one Conference to another, answer "no".
Do you want to automatically delete orphaned App\Entity\Commentaire objects
(orphanRemoval)? (yes/no) [no]:
> yes
updated: src/Entity/Conference.php
updated: src/Entity/Commentaire.php
```

Les entités après l'ajout de la relation :

```
#[ORM\Entity(repositoryClass: ConferenceRepository::class)]
class Conference
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 50)]
    private $ville;

    #[ORM\Column(type: 'integer')]
    private $annee;

    #[ORM\Column(type: 'boolean')]
    private $international;

    #[ORM\OneToMany(mappedBy: 'conference', targetEntity: Commentaire::class,
orphanRemoval: true)]
    private $comment;

    public function __construct()
    {
        $this->comment = new ArrayCollection();
    }

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

    public function getVille(): ?string
    {
        return $this->ville;
    }

    public function setVille(string $ville): self
    {
```

```
        $this->ville = $ville;

        return $this;
    }

    public function getAnnee(): ?int
    {
        return $this->annee;
    }

    public function setAnnee(int $annee): self
    {
        $this->annee = $annee;
        return $this;
    }

    public function getInternational(): ?bool
    {
        return $this->international;
    }

    public function setInternational(bool $international): self
    {
        $this->international = $international;
        return $this;
    }

    /**
     * @return Collection<int, Commentaire>
     */
    public function getComment(): Collection
    {
        return $this->comment;
    }

    public function addComment(Commentaire $comment): self
    {
        if (!$this->comment->contains($comment)) {
            $this->comment[] = $comment;
            $comment->setConference($this);
        }

        return $this;
    }

    public function removeComment(Commentaire $comment): self
    {
        if ($this->comment->removeElement($comment)) {
            // set the owning side to null (unless already changed)
            if ($comment->getConference() === $this) {
```

```
            $comment->setConference(null);
        }
    }
    return $this;
  }
}
```

```
#[ORM\Entity(repositoryClass: CommentaireRepository::class)]
class Commentaire
{
  #[ORM\Id]
  #[ORM\GeneratedValue]
  #[ORM\Column(type: 'integer')]
  private $id;

  #[ORM\Column(type: 'string', length: 50)]
  private $auteur;

  #[ORM\Column(type: 'text')]
  private $text;

  #[ORM\Column(type: 'string', length: 60)]
  private $email;

  #[ORM\Column(type: 'datetime')]
  private $dateCreat;

  #[ORM\ManyToOne(targetEntity: Conference::class, inversedBy: 'comment')]
  #[ORM\JoinColumn(nullable: false)]
  private $conference;

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

  public function getAuteur(): ?string
  {
    return $this->auteur;
  }

  public function setAuteur(string $auteur): self
  {
    $this->auteur = $auteur;
    return $this;
  }

  public function getText(): ?string
  {
```

```
        return $this->text;
    }

    public function setText(string $text): self
    {
        $this->text = $text;
        return $this;
    }

    public function getEmail(): ?string
    {
        return $this->email;
    }

    public function setEmail(string $email): self
    {
        $this->email = $email;
        return $this;
    }

    public function getDateCreat(): ?\DateTimeInterface
    {
        return $this->dateCreat;
    }

    public function setDateCreat(\DateTimeInterface $dateCreat): self
    {
        $this->dateCreat = $dateCreat;
        return $this;
    }

    public function getConference(): ?Conference
    {
        return $this->conference;
    }

    public function setConference(?Conference $conference): self
    {
        $this->conference = $conference;
        return $this;
    }
}
```

5. Ajouter des propriétés

Ajouter la propriété *tel* dans l'entité *Commentaire* :
Exécutez *make:entity* une autre fois et ajoutez une propriété/colonne tel de.
Si vous préférez ajouter de nouvelles propriétés manuellement, la commande *make:entity* peut générer les méthodes getter et setter pour vous :

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

6. Migration : créer le schéma de la base de données

Le modèle du projet est maintenant entièrement décrit par les deux classes générées.
Ensuite, nous devons créer les tables de la base de données liées à ces entités PHP :

```
symfony console make:migration ou php bin/console make:migration
symfony console doctrine:migrations:migrate
```

Pour générer les entités à partir des tables de la base de données :

```
php bin/console doctrine:mapping:import "App\Entity" annotation --path  src/Entity
php bin/console make:entity --regenerate App
```

Pour générer le *repository*, ajouter dans chaque entité :

```
* @ORM\Entity(repositoryClass="App\Repository\NomEntitéRepository")
```

puis :

```
php bin/console make:entity --regenerate App
```

7. Manipulation des données

a. Insertion

Vous allez ajouter une conférence.
Créez le controleur ConferenceController :

```
symfony console make:controller ConferenceController
```

Pour utiliser le « Validator » :

```
composer require symfony/validator
```

Dans le contrôleur, vous pouvez créer un nouvel objet Conference, y définir des données et l'enregistrer :

```
// src/Controller/ConferenceController.php
namespace App\Controller;
use App\Entity\Conference;
use Doctrine\Persistence\ManagerRegistry;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Validator\Validator\ValidatorInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
```

```
class ConferenceController extends AbstractController
{
   /**
    * @Route("/conference/ajout", name="ajout_conf")
    */
   public function createConference(ManagerRegistry $doctrine, ValidatorInterface
$validator): Response
   {
      $entityManager = $doctrine->getManager();

      $conf = new Conference();
      $conf->setVille('Medenine');
      $conf->setAnnee(2022);
      $conf->setInternational(true);

      //insérer les données
      $entityManager->persist($conf);
      $entityManager->flush();

       $errors = $validator->validate($conf);
       if ($errors ->count() > 0)
            return new Response((string) $errors, 400);

  else
      return new Response('Sauvegarde d'une conference avec id '.$conf->getId());
   }
}
```

b. Récupérer des données

Supposons que vous souhaitiez récupérer les informations d'une conférence ayant un id spécifique :

```
// src/Controller/ConferenceController.php
namespace App\Controller;

use App\Entity\Product;
use Symfony\Component\HttpFoundation\Response;
// ...

class ConferenceController extends AbstractController
{
  /**
    * @Route("/conference/{id}", name="afficher_conf")
    */
   public function afficher(ManagerRegistry $doctrine, int $id): Response
   {
      $conf = $doctrine->getRepository(Conference::class)->find($id);

      if (!$conf) {
        // throw $this->createNotFoundException(
```

```
            //   'La conference numéro '.$id. 'est inéxistante'
          //  );
          return new Response('La conférence numéro '.$id. ' est inéxistante',401);
        }

        return new Response('Lieu: '.$conf->getVille());

        // ou render un template
        // return $this->render('conference/affichage.html.twig', ['conference' => $conf]);
    }
}
```

Vous pouvez utiliser la classe *ConferenceRepository* comme suit :

```
$repository = $doctrine->getRepository(Conference::class);

// récupérer une conférence selon la clé primaire (id)
$conf = $repository->find($id);

// récupérer une conférence selon la ville
$conf = $repository->findOneBy(['ville' => 'Medenine']);
// ou selon la ville et l'année
$confs= $repository->findOneBy([
   'ville' => 'Medenine',
   'annee' => 2021,
]);

// récupérer les conférences d'une ville spécifique ordonnées selon l'année
$confs = $repository->findBy(
   ['ville' => 'Medenine'],
   ['annee' => 'ASC']
);

// récupérer toutes les conférences
$confs = $repository->findAll();    }
}
```

c. Mise à jour

Les étapes de mise à jour :

- Récupérer l'objet
- Effectuer la modification désirer
- Mettre à jour la table

```
// src/Controller/ConferenceController.php
namespace App\Controller;

use App\Entity\Conference;
use App\Repository\ConferenceRepository;
use Symfony\Component\HttpFoundation\Response;
// ...
```

```
class ConferenceController extends AbstractController
{
    /**
     * @Route("/conference/editer/{id}")
     */
    public function maj(ManagerRegistry $doctrine, int $id): Response
    {
        $entityManager = $doctrine->getManager();
        $conf = $entityManager->getRepository(Conference::class)->find($id);

        if (!$conf) {
            throw $this->createNotFoundException(
                'Conférence introuvable '.$id
            );
        }

        $conf->setVille('Tunis');
        $entityManager->flush();

        return $this->redirectToRoute('afficher_conf', [
            'id' => $conf->getId()
        ]);
    }
}
```

d. Suppression

La suppression d'un objet est similaire à la mise à jour, mais nécessite un appel à la méthode *remove()* :

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

e. Les requêtes

On peut définir toutes les requêtes nécessaires dans la classe Repository, avec plusieurs manières.
Supposons qu'on veut récupérer toutes les conférences qui ont eu lieu après une années spécifique :

```
// src/Repository/ConferenceRepository.php

// ...
class ConferenceRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Conference::class);
    }

    /**
```

```
     * @return Conference[]
     */
    public function confApresAnnee(int $annee): array
    {
        $entityManager = $this->getEntityManager();

        $query = $entityManager->createQuery(
            'SELECT c
            FROM App\Entity\Conference c
            WHERE c.annee > :an
            ORDER BY c.annee ASC'
        )->setParameter('an', $annee);

        // retourner les conférences
        return $query->getResult();
    }
}
```

ou :

```
// src/Repository/ConferenceRepository.php
// ...
class ConferenceRepository extends ServiceEntityRepository
{
    public function confApresAnnee(int $annee, bool $includeUnavailableConferences = false): array
    {
        //  "c" est un alias
        $qb = $this->createQueryBuilder('c')
            ->where('c.annee > :an')
            ->setParameter('an', $annee)
            ->orderBy('c.annee', 'ASC');

        if (!$includeUnavailableConferences) {
            $qb->andWhere('c.available = TRUE');
        }
        $query = $qb->getQuery();

        return $query->execute();
        // si vous voulez une seule conférence :
        // $conf = $query->setMaxResults(1)->getOneOrNullResult();
    }
}
```

ou :

```
// src/Repository/ConferenceRepository.php
// ...
class ConferenceRepository extends ServiceEntityRepository
{
```

```
    public function confApresAnnee(int $annee){
        $conn = $this->getEntityManager()->getConnection();

        $sql = '
            SELECT * FROM conference c
            WHERE c.annee > :an
            ORDER BY c.annee ASC
            ';
        $stmt = $conn->prepare($sql);
        $resultSet = $stmt->executeQuery(['an' => $annee]);
        return $resultSet->fetchAllAssociative();
    }
}
```

Ensuite vous pouvez utiliser la méthode de Repository, dans le contrôleur, comme suit :

```
public function confsAnnee(ConferenceRepository $rep)
    {
        $annee =2020;
        $confs = $rep->confApresAnnee($annee);
        return $this->render('conference/affichage.html.twig', ['conferences' => $confs]);
    }
```

Application :

1. Ajouter deux commentaires à la base de données.
2. Modifier l'auteur du premier commentaire.
3. Supprimer le deuxième commentaire.