Doctrine :

Partie 1 : Premiers pas

Fil directeur de cette présentation

Création d'une « todo liste » en 4 heures

Pré-requis

Savoir installer et créer des contrôleurs Symfony

Connaître la syntaxeTwig

Maîtriser la programmation orientée Objet : instanciation, héritage, getters/setters, méthodes magiques et Composer

Code source public

https://github.com/AlexisRomeroDev/DoctrineTutoral

1. Installation de Doctrine et création BDD

Installation de Doctrine

https://symfony.com/doc/current/doctrine.html

```
composer require symfony/orm-pack
```

Créer la base de données

Puis renseigner la configuration à la base de données dans le fichier .env

Plusieurs drivers sont possibles (mysql, sqlite, postgresql). Exemple avec Maria-db :

```
DATABASE_URL="mysql://login:password@127.0.0.1:3306/db_name?serverVersion=10.3.34-MariaDB"
```

Pour connaître votre version de serveur Mysql, executer cette commande :

```
mysql -V
```

Créer la base de données

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

Installation du makerBundle

https://symfony.com/bundles/SymfonyMakerBundle/current/index.html

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

2. Création d’une entité Doctrine

Générer une entité

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

Création d’une entité Todo

Analyse des fichiers générés

```
git status
```

src/Entity/Todo.php

L’entité Todo est une classe php.
Se référer au cours sur la POO

L’entité Todo comporte des annotations :
@ORM\Entity( )
@ORM\Column()
Ces annotations définissent le modèle physique de données. Une entité correspond à une table en base de données et chaque propriété est une colonne de cette table.
Tout savoir sur les annotations :
https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/reference/annotations-reference.html
Il existe des formats alternatifs aux annotations :Par ex, les formats XML et YAML

src/Repository/Todo.php

Le repository centralise tout ce qui touche à la récupération des entités en base de données.

Migrations

Pour créer ce schéma en base de données : 2 commandes :

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

Modifier une entité

Ajouter un champs « name »

Relancer les commandes :

```
bin/console make:entity
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

3. CRUD

Pour notre entité, on va vouloir faire des opérations de base : Ajouter des todo, les afficher, les modifier et les supprimer.

Installer les bundles

Si nécéssaire :

```
composer require form validator twig-bundle security-csrf annotations
```

Générer un Crud

```
bin/console make:crud
```

Tester dans un navigateur : http://127.0.0.1:8000/todo/

Fichiers créés :

```
created: src/Controller/TodoController.php
created: src/Form/TodoType.php
created: templates/todo/_delete_form.html.twig
created: templates/todo/_form.html.twig
created: templates/todo/edit.html.twig
created: templates/todo/index.html.twig
created: templates/todo/new.html.twig
created: templates/todo/show.html.twig
```

Analyse des modifications dans le contrôleur

Les routes

=> Définies dans les annotations

Les requêtes en base de données

Dans la méthode index(), qui a vocation a retourner la liste des todos :

```
$todoRepository-> findAll()
```

Cette méthode récupère tous les todos actuellement en base de données. Cette méthode est disponible par héritage de la classe lib/Doctrine/ORM/EntityRepository.php. Quelques méthodes disponibles :

```
• findAll()
• find($id)
• findBy()
• findOneBy()
• ...
```

Le paramConverter

Dans la méthode show(), qui a vocation à retourner le todo dont l'id est passé en paramètre d'url, pas de méthode « magique ». Le todo est récupéré grace au mécanisme de ParamConterter

Pour en saoir plus sur les ParamConverter :
https://symfony.com/bundles/SensioFrameworkExtraBundle/current/annotations/converters.html

Dans les méthode edit(), new() et delete() :

```
$entityManager->persist($todo);
$entityManager->remove($todo);
$entityManager->flush();
```

Ce sont les méthodes traditionnelles pour insérer, modifier et supprimer des données en base.

Noter l'utilisation de la classe src/Form/TodoType.php qui permet de créer un formulaire HTML correspondant à l'entité en Base de donnée.
1 Entité = 1 FormType

Modifier une entité existante

On souhaite ajouter une description à chaque Todo

Modification de l'entité

La modification d'une entité, comme la création d'ailleurs, peut se faire manuellement, ou bien par la commande

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

Cette commande ajoute une nouvelle propriété à la classe TODO

```
@ORM\Column(type :"text", nullable :true)
private?string $description = null ;
```

ainsi que les getters et les setters.

Modification du formulaire

Il faut, dans tous les cas, compléter la classe TodoType

```
$builder->add('description');
```

Modification du template

```
{{ todo.description }}
```

4. Fixtures

Deux bundles bien utiles :

```
composer require zenstruck/foundry --dev
```

https://symfony.com/bundles/ZenstruckFoundryBundle/current/index.html

```
composer require --dev orm-fixtures
```

Rédaction des fixtures & chargement en base de données

Consulter la documentation :

https://symfony.com/bundles/DoctrineFixturesBundle/current/index.html

https://aymeric-cucherousset.fr/symfony-6-fixtures/

Travaux pratiques

Créer un CRUD avec un jeu de données pour une entité Post qui contient les champs Titre, Contenu et Date