IUT La Rochelle

Programmation Web côté serveur

2ème année DUT

2020-2021

Module **PWS3**

Découverte symfony suite la couche métier

Reprenez votre projet Jeuxdicode débuté avant les congés et mettez en place votre environnement de développement.

Prérequis : avoir terminé le TD précédent de Symfony

Si ce n'est pas le cas terminer d'abord le TD précédent Symfony.

Etape 1 : La couche métier, les entités

Notion d'ORM : Object Relation Mapper, technique permettant de faire le lien avec la base de données relationnelle en définissant des liens entre les tables de la base et les objets (entités) du modèle. L'ORM par défaut de Symfony s'appelle doctrine 2.

https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/index.html

Dans un ORM, un objet métier dont vous voulez gérer la persistance s'appellera une entité.

Paramétrage de la BD :

Il nous faut dans un premier temps paramétrer notre projet Symfony pour faire le lien avec la base de données choisie.

Pour cela nous allons créer un fichier .env.local à partir d'une copie du fichier **.env (notez que ce fichier .env.local est déjà prévu dans le .gitignore à la racine de votre projet symfony d'après vous pourquoi?)**

Le host sera `iutlr-info-mysql8-demo`

Notre base de données s'appellera **dbdemo**.

Il faut mettre à jour le login et mdp (**demo** / **demo**).

Remplacez :

DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7

Par

DATABASE_URL=mysql://demo:demo@iutlr-info-mysql8-demo/dbdemo?serverVersion=5.7

Rappel : Lancer le terminal de votre container docker app, pour exécuter les commandes php.

docker-compose exec app bash

Et positionnez-vous dans le répertoire jeuxdicode

Création de la base de données

Quand on souhaite créer une base de données via doctrine, on pourrait utiliser le générateur de symfony, et la commande suivante, mais attention nous n'en avons pas besoin.

php bin/console doctrine:database:create

Souvenez-vous, vous disposez déjà au lancement de votre container bd, d'une base de données dbdemo, que nous allons utiliser.

Rappel sous phpstorm, ajoutez la source de données mysql dbdemo.

Création d'une entité :

Une entité est, du point de vue PHP, un simple objet. Du point de vue de doctrine c'est un objet complété avec des informations de mapping qui lui permettent d'enregistrer l'objet dans une base.

Nous allons mettre en place une première entité, l'entité Evenement :

Pour cela nous allons utiliser le générateur de symfony.

php bin/console make:entity

Nom de l'entité : Evenement

Doctrine2 ajoutera automatiquement l'id, donc débuter la création des attributs au titre.

titre => string [60] not null

date_heure_debut => datetime not null

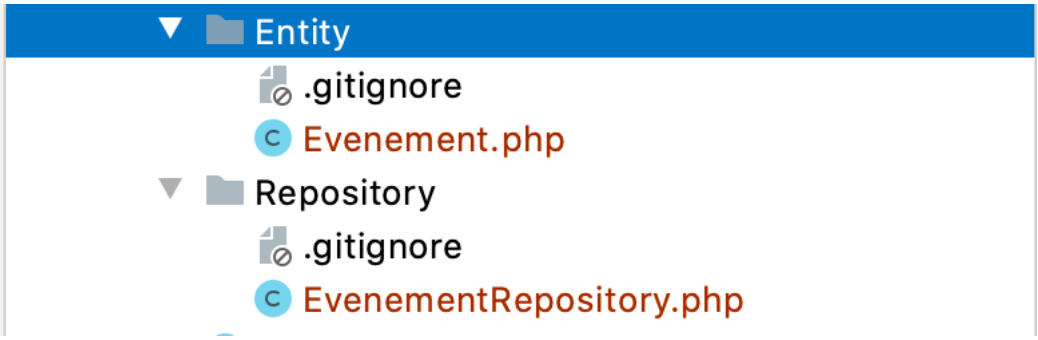
date_heure_debut => datetime not null

nbpartmax => integer null

adresse => string [150] not null

```
root@4d25542640e1:/var/www/html/jeuxdicode# php bin/console make:entity

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

 created: src/Entity/Evenement.php
 created: src/Repository/EvenementRepository.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):
 > titre

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

 Field length [255]:
 > 60

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

 updated: src/Entity/Evenement.php

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

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

Observons d'abord ce qui vient d'être généré :



Un répertoire Entity et un répertoire Repository a été ajouté.

Observons la classe Événement :

L'annotation **Entity** s'applique à la classe, elle est placée avant la définition de la classe. Elle définit la classe comme étant une entité et doctrine la fera persister.

Il existe un paramètre facultatif pour cette annotation, **repositoryClass**.

Il gère l'espace de nom complet du repository qui gère cette entité. (Nous reviendrons plus loin sur cette notion de repository)

L'annotation **Table** (non présente ici) s'applique sur une classe également. Cette annotation est facultative, elle sert à personnaliser le nom de la table dans la base de données.

Par défaut ce nom est le même que l'entité.

Observons maintenant le contenu de la classe Evenement.

```php
<?php

namespace App\Entity;

use App\Repository\EvenementRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=EvenementRepository::class)
 */
class Evenement
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=60)
     */
    private $titre;
```

L'annotation Column, elle s'applique sur un attribut de classe, elle se positionne juste avant la définition PHP de l'attribut. Cette annotation définit les caractéristiques de la colonne concernée.

On peut noter qu'un attribut a été généré automatiquement, il s'agit de Id qui correspondra à la clé primaire côté BD.

Les types définis sont des types doctrine, ils font la transition des types SQL aux types PHP.

On peut noter qu'un attribut a été généré automatiquement, il s'agit de Id qui correspondra à la clé primaire côté BD.

Les types définis sont des types doctrine, ils font la transition des types SQL aux types PHP.

On pourrait utiliser cet objet dès maintenant, du côté PHP une entité est un objet :

```php
<?php

namespace App\Controller;

use App\Entity\Evenement;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class IndexController extends AbstractController
{
    /**
     * @Route("/", name="home")
     */
    public function index()
    {
        $evt=new Evenement();
        $evt->setTitre('présentation du framework symfony');
        return $this->render('index/index.html.twig', [
            'controller_name' => 'IndexController',
            'evt'=> $evt,
        ]);
    }
}
```

Et côté de la vue :

```
<div class="col-md-4">
    <H2>Prochain évènement</H2>
    <p>{{ evt.titre }}</p>
    <p><a class="btn btn-success pull-right" href=""> Détail évènement
</a></p>
</div>
```

Ce sont les annotations qui permettent de faire le lien avec la BD.

Une entité est un objet du point de vue de PHP et du point de vue de doctrine un objet complété avec des informations de mapping qui lui permettent d'enregistrer correctement l'objet en base.