Module **PWS3** **Symfony ORM 3**

Remplir la base de données avec une fixture doctrine

Les fixtures doctrine remplissent la BD avec un jeu de données que nous allons définir. Cela permet de tester le fonctionnement de notre application avec un jeu d'essai réaliste.

Pour cela nous allons installer un bundle avec le gestionnaire de dépendance composer.

Nous allons installer le bundle DoctrineFixtureBundle pour cela nous utilisons la recette flex orm-fixtures.

```
composer require orm-fixtures
```

Les fixtures se trouvent dans le répertoire DataFixtures qui a été ajouté à l'installation. Un fichier AppFixtures.php est également généré, il contient le début de la classe AppFixtures qui étend la classes Fixtures.

Voici un exemple avec 2 évènements dont un évènement contenant une session et 2 participants.

```php
<?php

namespace App\DataFixtures;

use App\Entity\Evenement;
use App\Entity\Membre;
use App\Entity\Session;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        // $product = new Product();
        // $manager->persist($product);
        //Les membres
        $membre1=new Membre();
        $membre1->setNom( nom: 'dupont');
        $membre1->setPrenom( prenom: 'paul');
        $membre1->setTel( tel: '06xxxxxxxx');
        $membre1->setAdressemail( adressemail: 'dupontp@test.com');
        $manager->persist($membre1);
        $membre2=new Membre();
        $membre2->setNom( nom: 'durand');
        $membre2->setPrenom( prenom: 'pierre');
        $membre2->setTel( tel: '06xxxxxxxx');
        $membre2->setAdressemail( adressemail: 'durandp@test.com');
        $manager->persist($membre2);
```

```php
        //Les évènements
        $event1=new Evenement();
        $event1->setTitre( titre: 'intro symfony');
        $DateD='10/12/2020';
        $DateF='10/12/2020';
        $event1->setDateHeureDebut(new \DateTime($DateD));
        $event1->setDateHeureFin(new \DateTime($DateF));
        $event1-> setNbpartmax( nbpartmax: 9);
        $event1->setAdresse( adresse: 'en ligne');
        $event1->setEvtcourant( evtcourant: false);

        $event1->addMembre($membre1);
        $event1->addMembre($membre2);
        $manager->persist($event1);

        $event2=new Evenement();
        $event2->setTitre( titre: 'intro docker');
        $DateD='9/12/2020';
        $DateF='9/12/2020';
        $event2->setDateHeureDebut(new \DateTime($DateD));
        $event2->setDateHeureFin(new \DateTime($DateF));
        $event2-> setNbpartmax( nbpartmax: 9);
        $event2->setAdresse( adresse: 'en ligne');
        $event2->setEvtcourant( evtcourant: false);

        $event2->addMembre($membre1);
        $manager->persist($event2);
```

```php
        //Les sessions
        $session1=new Session();
        $session1->setTitre( titre: 'intro symfony les bases');
        $session1->setNomauteur( nomauteur: 'CF');
        $session1->setEvenement($event1);
        $manager->persist($session1);

        $manager->flush();
    }
}
```

Complétez votre AppFixtures.php avec votre jeu d'essai permettant de tester la liste des évènements et le détail d'un évènement avec la ou les sessions et les membres participants.
Explications persist() et flush() slide 14 du support ORM-Doctrine

Dans la console pour utiliser la fixture :
php bin/console doctrine:fixtures:load
Testez que votre page detailevt pour l'id de votre événement test (voir l'id dans phpstorm, comme il est autoincrémenté ce n'est plus 1) est toujours opérationnelle.

Le rôle des repositories

Un repository centralise tout ce qui touche à la récupération de vos entités.
Il existe un repository par entité. Cela n'empêchera pas un repository d'utiliser plusieurs entités dans le cas d'une jointure par exemple.

Avec doctrine il y a deux façons de récupérer les entités :

- En Doctrine Query Language (DQL) : du SQL adapté à la vision objet utilisée par doctrine
- Avec le Query Builder: sert à construire une requête par étape

Les méthodes de récupération de base :

Méthode	description
find ($id)	Récupère l'entité correspondant à l'id $id Si aucune entité ne correspond à cette id elle retourne null
findAll()	Retourne toutes les entités contenues dans la base de données. Le format de retour est un array
findBy()	Elle retourne une liste d'entités avec possibilité d'appliquer un filtre pour ne retourner que celles correspondant à un ou plusieurs critères .$repository = $this->getDoctrine()->getRepository('AppBundle:Product'); // query for multiple products matching the given name, ordered by price $products = $repository->findBy(array('name' => 'Keyboard'), array('price' => 'ASC')); Si aucune entité ne correspond à cette id elle retourne null.
findOneBy()	Même principe que FindBy() mais elle ne retourne qu'une seule entité. $repository = $this->getDoctrine()->getRepository('AppBundle:Product'); // query for a single product matching the given name and price $product = $repository->findOneBy(array('name' => 'Keyboard', 'price' => 19.99)); Si aucune entité ne correspond à cette id elle retourne null.

Les méthodes magiques :

Méthode	description
findByX($valeur)	Il faut remplacer le X par le nom d'une propriété de l'entité.
findOneByX($valeur)	Il faut remplacer le X par le nom d'une propriété de l'entité.

Personnalisation de requête avec le QueryBuilder :

3 types d'objet vont servir :
- Le queryBuilder
- Le query
- Le résultat

Le QueryBuilder permet de construire une query :
http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html

Pour récupérer un queryBuilder nous allons utiliser le gestionnaire d'entités fourni par le framework, on y accède depuis le repository par la variable _em
Le code complet pour récupérer un queryBuilder depuis une méthode :
$this->_em->createQueryBuilder()
Mais cette méthode retournera un queryBuilder vide.
On utilisera une méthode directe du repository CreateQueryBuilder($alias) qui utilise la méthode du gestionnaire d'entité en définissant le select et le where.
L'alias en argument de la méthode est le raccourci de l'entité du repository, l'usage veut que l'on utilise la première lettre de l'entité comme alias.

Nous allons recréer la méthode findAll() sur le repository Evenement :

```
public function myFindAll(){
  //création du queryBuilder paramétré avec l'alias de l'entité événement
  $queryBuilder=$this->createQueryBuilder('e');
  //récupération de la query à partir du queryBuilder
  $query=$queryBuilder->getQuery();
  //récupération du résultat à partir de la query
  $results=$query->getResult();
  //on retourne les résultats
  return $results;
}
```

Mettez en place cette méthode et modifiez le contrôleur pour obtenir la liste des événements en utilisant cette méthode du repository.

Consultez le profiler pour voir la requête qui a été exécutée.

Nous souhaitons maintenant avoir sur la route /listevt 2 blocs, la liste de tous les évènements et la liste contenant uniquement les événements de 2020.
Mettez à jour votre jeu d'essai et relancez vos fixtures pour disposer d'un domaine valide et d'un domaine invalide pour votre requête.
Nous pouvons définir des paramètres avec :nom_du_paramètre, puis on lui attribue une valeur avec setParameter('nom_du_paramètre',$valeur)

```
$querybuilder=$this->createQueryBuilder( alias: 'e');
//récupération de la query à partir du querybuilder
$query=$querybuilder->where( predicates: 'e.date_heure_debut between :start and :end')
                    ->setParameter( key: 'start',new \DateTime( time: '2020-01-01 00:00:00'))
                    ->setParameter( key: 'end',new \DateTime( time: '2020-12-31 23:59:00'))
                    ->getQuery();
```

Mettez en place la méthode myFindAll2020() dans votre repository et modifiez le contrôleur Evenement et la vue listevt.html.twig pour obtenir la liste des événements de 2020 en utilisant cette nouvelle méthode.

Regardons de plus près la query.
La query est l'objet à partir duquel on extrait les résultats.Il y a plusieurs façon d'extraire les résultats de la requête.

Méthode	description
getResult()	Cette méthode exécute la requête et retourne un tableau contenant les résultats sous forme d'objets. Même si la requête ne renvoie qu'un seul résultat, elle retourne un tableau
getSingleResult()	Cette méthode exécute la requête et retourne un seul résultat. Et déclenche une exception si il n'y a pas de résultat
...	http://docs.doctrine-project.org/en/latest/reference/query-builder.html#executing-a-query

Personnalisation de requête avec le DQL :
https://symfony.com/doc/current/doctrine.html

DQL est une sorte de SQL adapté à l'ORM doctrine. On écrit la requête en chaîne de caractères mais avec une vision objet et non table.
Exemple récupération de la liste des événements en DQL:

```
$query=$this->_em->createQuery('SELECT e FROM App\Entity\Evenement e');
```

Mettez en place la méthode myFindAllDQL() dans votre repository et modifiez le contrôleur pour obtenir la liste des événements en utilisant cette nouvelle méthode.

Nous souhaitons maintenant écrire une requête en DQL permettant d'afficher uniquement les événements dont le titre contient un mot clé par exemple symfony. Pour cela :

- Mettez à jour votre fixture événement pour créer le jeu d'essai adapté
- Ajoutez une méthode myFindTitreDQL() dans le repository pour fournir le résultat de cette requête

```php
public function MyFindTitreDQL($titre):array
{
    $query=$this->_em->createQuery('SELECT e FROM App\Entity\Evenement e Where e.titre like :titre')
        ->setParameter( key: 'titre', value: '%'.$titre.'%');
    // on retourne les résultats
    return $query->getResult();

}
```

Astuce : le %% du Like permettant de dire contient se rajoute dans le setParameter

- Mettez à jour le contrôleur Evenement et l'action listevt pour qu'elle affiche le résultat de la requête.

Pour Aller plus loin :

Vous pouvez tester les autres méthodes magiques, de nouvelles requêtes en DQL ou avec le QueryBuilder pour tester les potentialités de vos repositories.