SIN6U5 : Développement web 2 (Prog. web coté serveur) L'ORM Doctrine

Luigi Santocanale
LIF, Aix-Marseille Université

15 mars 2018

Plan

Plan

Documentationn de Doctrine

- Documentation du projet Doctrine
- Didacticiels
- Guide référence
- Depuis la doc de Symfony

Présentation de Doctrine

Historique :

- Lancé en 2006 par Konsta Vesterinen, le projet Doctrine est inspiré de l'ORM Java Hibernate2 et du patron de conception active record en Ruby.
- On est à la version 2.6 (decembre 2017).

Projet indépendant de Symfony : peut être utilisé avec d'autres frameworks.

Architecture :

- DBAL (DataBase Abtraction Layer) : couche au dessus de PDO
- ORM (Object Relational Mapping) : interface qui permet de faire le lien ou "mapping" entre nos objets et les éléments de la base de données. Ainsi un enregistrement correspondra à une instance (et vice et versa) et une table correspond à une classe (ou entité).

Éléments

Notion importantes :

- Entity : objet d'une classe susceptible de persistance.
- EntityManager : le factotum ! ! !
- Repository : le conteneur des entités d'une même classe.
- UnitOfWork : les esclaves de l'EntityManager.

Une entité est dans un de ces états :

- NEW (NOUVELLE) : pas de persistance, pas associé à l'EntityManager et à l'UnitOfWork.
- MANAGED : persistante, associée à l'EntityManager.
- DETACHED (detachée) : persistante, pas associée à l'EntityManager et à l'UnitOfWork.
- REMOVED : persistante, associée à l'EntityManager, vas être détruite (de la base de données) à la fin de la prochaine transaction.

Paramétrage de Doctrine

app/config/config.yml :

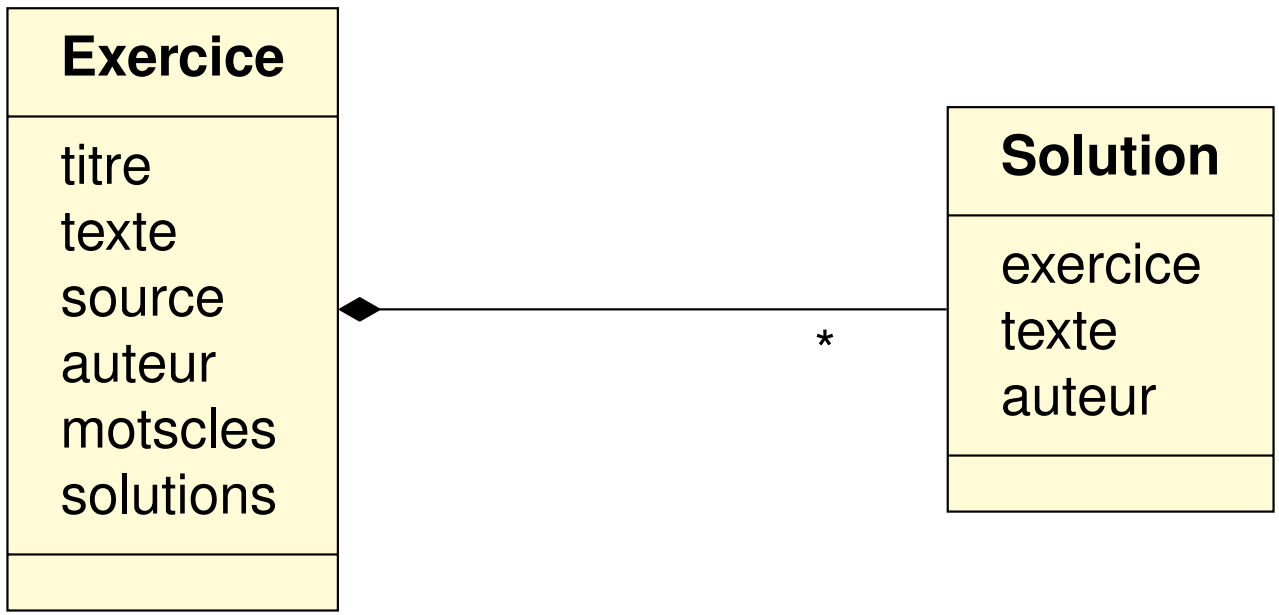
```
# Doctrine Configuration
doctrine:
    dbal:
        driver: pdo_sqlite
        charset: UTF8
        path: '%kernel.project_dir%/var/db/database.sqlite'

    orm:
        auto_generate_proxy_classes: '%kernel.debug%'
        naming_strategy: doctrine.orm.naming_strategy.underscore
        auto_mapping: true
```

Bien-sur, il faut créer le répertoire `var/db/` !!!

Plan

Le modèle, en objet



Le modèle, en PHP

Ces classes dans le répertoire `src/AppBundle/Entity/`

```php
<?php
class Exercice {
    private $titre;
    private $texte;
    private $source;
    private $auteur;
    private $motscles;
    private $solutions;
}
```

```php
<?php
class Solution {
    private $exercice;
    private $texte;
    private $auteur;
}
```

Le modèle, en PHP

Ajout des champs `id` :

```php
<?php
class Exercice {
    private $id;

    private $titre;
    private $texte;
    private $source;
    private $auteur;
    private $motscles;
    private $solutions;
}
```

```php
<?php
class Solution {
    private $id;

    private $exercice;
    private $texte;
    private $auteur;
}
```

. . . plus ajout de « setters » et « getters ».

De PHP à Doctrine

Ajout des annotations de « mapping » :

```php
<?php
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="exercice")
 */
class Exercice {
    /**
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     **/
    private $id;

     /**
     * @ORM\Column(name="titre", type="string")
     **/
    private $titre='';

     /**
     * @ORM\Column(name="texte", type="string")
     **/
    private $texte='';
```

De PHP à Doctrine (II)

```
    /**
     * @ORM\Column(name="source", type="string")
     **/
    private $source='';

    /**
     * @ORM\Column(name="auteur", type="string")
     **/
    private $auteur='';

    /**
     * @ORM\Column(name="motscles", type="string")
     **/
    private $motscles='';

    /**
     * @ORM\Column(name="solutions", type="string")
     **/
    private $solutions=''; // Annotation à modifier
}
```

De PHP à Doctrine (III)

```php
<?php
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="solution")
 */
class Solution {
    /**
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     **/
    private $id;

    /**
     * @ORM\Column(name="exercice", type="string")
     **/
    private $exercice=''; // Annotation à modifier

    /**
     * @ORM\Column(name="texte", type="string")
     **/
    private $texte='';

     /**
     * @ORM\Column(name="auteur", type="string")
     **/
    private $auteur='';
}
```

Génération des entités en console

Pour ajouter les « setters/getters » on peut utilser :

```
php bin/console doctrine:generate:entities AppBundle
```

Au lieu de créer les entités à la main, on peut utilser `bin/console` :

```
php bin/console doctrine:generate:entity
```

Création de la base de données

Création de la base :

```
php bin/console doctrine:database:drop --force
php bin/console doctrine:database:create
```

Validation du schéma de traduction :

```
php bin/console doctrine:schema:validate
```

ou mise à jour du schema :

```
php bin/console doctrine:schema:update --force
```

Code sql engendré

```
CREATE TABLE exercice (
        id INTEGER NOT NULL,
        titre VARCHAR(255) NOT NULL,
        texte VARCHAR(255) NOT NULL,
        source VARCHAR(255) NOT NULL,
        auteur VARCHAR(255) NOT NULL,
        motscles VARCHAR(255) NOT NULL,
        solutions VARCHAR(255) NOT NULL,
        PRIMARY KEY(id)
        );

CREATE TABLE solution (
        id INTEGER NOT NULL,
        exercice VARCHAR(255) NOT NULL,
        texte VARCHAR(255) NOT NULL,
        auteur VARCHAR(255) NOT NULL,
        PRIMARY KEY(id)
        );
```

Ajout d'une route `test`

```php
<?php

// src/AppBundle/Controller/TestController.php

namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use AppBundle\Entity\Exercice;
use Symfony\Component\HttpFoundation\Response;
use Doctrine\ORM\EntityManagerInterface;

class TestController extends Controller {

    /**
     * @Route("/test", name="test")
     */
    public function testAction() {
        $entityManager = $this->getDoctrine()->getManager();

        $exercice = new Exercice();
        $exercice->setTitre('Test');
        $exercice->setTexte('Exo de test');

        $entityManager->persist($exercice);
        $entityManager->flush();

        return new Response('L\'id du nouvel exercice est : ' .
 ↪   $exercice->getId());
    }

}
```

Compléments

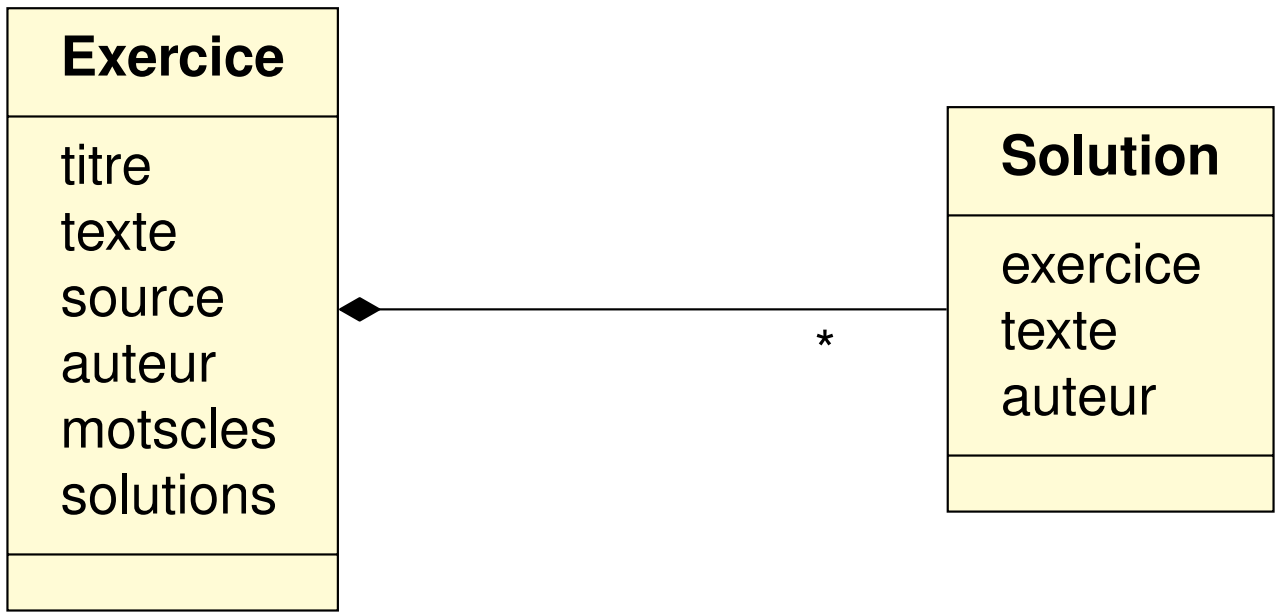
La doc de Doctrine sur les types

Observez la multitude des drivers :

`ibm_db2`, `pdo_sqlsrv`, `pdo_mysql`, `pdo_pgsql`, `pdo_sqlite`

Ajout de la relation OneToMany

Ajout de la relation OneToMany



Le coté « One »

Avant :

```
    /**
     * @ORM\Column(name="solutions", type="string")
     **/
    private $solutions=''; // Annotation à modifier
```

Après :

```
use Doctrine\Common\Collections\ArrayCollection;

    /**
     *  Un  exercice a plusieurs solutions
     * @ORM\OneToMany(targetEntity="Solution", mappedBy="exercice")
     */
    private $solutions;

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

Le coté « Many »

Avant :

```
/**
 * @ORM\Column(name="exercice", type="string")
 **/
private $exercice=''; // Annotation à modifier
```

Après :

```
/**
 *  Plusieur solutions pour un seul exercice
 * @ORM\ManyToOne(targetEntity="Exercice", inversedBy="solutions")
 */
private $exercice;

function __construct(Exercice $exercice){
    $this->exercice = $exercice;
    $exercice->getSolutions()->add($this);
}
```

Code sql engendré

```sql
CREATE TABLE exercice (
        id INTEGER NOT NULL,
        titre VARCHAR(255) NOT NULL COLLATE BINARY,
        texte VARCHAR(255) NOT NULL COLLATE BINARY,
        source VARCHAR(255) NOT NULL COLLATE BINARY,
        auteur VARCHAR(255) NOT NULL COLLATE BINARY,
        motscles VARCHAR(255) NOT NULL COLLATE BINARY,
        PRIMARY KEY(id)
        );

CREATE TABLE solution (
        id INTEGER NOT NULL,
        exercice_id INTEGER DEFAULT NULL,
        texte VARCHAR(255) NOT NULL COLLATE BINARY,
        auteur VARCHAR(255) NOT NULL COLLATE BINARY,
        PRIMARY KEY(id),
        CONSTRAINT FK_9F3329DB89D40298 FOREIGN KEY (exercice_id)
                        REFERENCES exercice (id)
                NOT DEFERRABLE INITIALLY IMMEDIATE
                    );

CREATE INDEX IDX_9F3329DB89D40298 ON solution (exercice_id);
```

Route pour tester

```php
class TestController2 extends Controller {

    /**
     * @Route("/test2", name="test2")
     */
    public function test2Action() {
        $entityManager = $this->getDoctrine()->getManager();

        $exercice = new Exercice();
        $exercice->setTitre('Test');
        $exercice->setTexte('Exo de test');
        $entityManager->persist($exercice);
        $entityManager->flush();

        $solution = new Solution($exercice);
        $solution->setTexte("Solution de l'exo précèdent" );
        $entityManager->persist($solution);
        $entityManager->flush();

        $msg1= sprintf("La solution %d est de l'exo %d\n",
                       $solution->getId(),
                       $solution->getExercice()->getId());
        $getId = function(Solution $solution){
            return strval($solution->getId());
        };
        $msg2= sprintf("L'exo %d a les solutions : %s\n",
                        $exercice->getId(),
                        implode(',',
                                $exercice->getSolutions()
                                ->map($getId)
                                ->toArray()));
        return new Response($msg1 . "<br />" .$msg2);
    }
}
```

La doc Doctrine sur les associations

Plan

Les repositoires

- Pour chaque type entité d'entité il y a une repositoire associée.
- Un repositoire est comme SGBD (en objet) auquel on demande de faire des recherches.
- API de recherche uniforme pour toutes les entités :

```
$result = $repository->find[One]ByChamp($value)
```

- On peut étendre la collection des recherches en utilsant PHP, DQL, ou SQL.
- Pour ce faire, on étends la classe

```
Doctrine\ORM\EntityRepository
```

- Ces extensions de classe doivent se trouver dans

```
usr/AppBundle/Repository/
```

Exemple : ajout d'une nouvelle route

```
class TestController3 extends Controller {

    /**
     * @Route("/test3/{titre}", name="test3")
     */
    public function test2Action($titre) {
        $repository = $this->getDoctrine()
                    ->getRepository(Exercice::class);
        $results=$repository->findByTitre($titre);
        $msg=$results?
            'Ids des résultats : '. $this->toString($results)
            :'Pas de résultats pour cette recherche';
        return new Response($msg);
    }

    private function toString($results){
        $strings = array_map(
            function($object){
                return strval($object->getId());
            }
            ,
            $results);
        return implode(', ',$strings);
    }
}
```

Espaces de noms utilisés

```php
<?php

// src/AppBundle/Controller/TestController3.php

namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use AppBundle\Entity\Exercice;
use Symfony\Component\HttpFoundation\Response;
use Doctrine\ORM\EntityRepository;
```

Les methodes de la classe `EntityRepository`

La classe `EntityRepository` propose par défaut quelques méthodes pour vous éviter de les écrire.

- `find` : prend un unique paramètre et recherche l'argument dans la clé primaire de l'entité.
- `findBy` : prend 4 paramètres (`$criteria`, `$orderBy`, `$limit`, `$offset`). Cette méthode retourne des résultats correspondant aux valeurs des clés demandées.
- `findAll` : alias de `findBy([])`. Il retourne par conséquent tous les résultats.
- `findOneBy` : fonctionne comme la méthode `findBy` mais retourne un unique résultat et non pas un tableau.

Source ici, un article intéressant, en français, sur les requetes avec Doctrine.

Recherche par mot-clé

Attention à la ligne 20 :

```
12  class TestController4 extends Controller {
13
14      /**
15       * @Route("/test4/{cle}", name="test4")
16       */
17      public function test2Action($cle) {
18          $repository = $this->getDoctrine()
19                      ->getRepository(Exercice::class);
20          $results=$repository->findWithMotCleDQL($cle);
21          $msg=$results?
22              'Ids des résultats : '. $this->toString($results)
23              :'Pas de résultats pour cette recherche';
24          return new Response($msg);
25      }
26
27      private function toString($results){

35      }
36  }
```

Recherche par mot-clé

```
1   <?php
2   namespace AppBundle\Entity;
3
4   use Doctrine\ORM\Mapping as ORM;
5   use Doctrine\Common\Collections\ArrayCollection;
6
7   /**
8    * @ORM\Entity(repositoryClass="AppBundle\Repository\ExerciceRepository")
9    * @ORM\Table(name="exercice")
10   */
11  class Exercice {

53  }
```

Mon repositoire

Avec SQL :

```php
<?php

// src/AppBundle/Repository/ExerciceRepository.php
namespace AppBundle\Repository;

use Doctrine\ORM\EntityRepository;

class ExerciceRepository extends EntityRepository
{
    public function findWithMotCleSQL($cle){
        $conn = $this->getEntityManager()->getConnection();
        $sql = '
        SELECT * FROM exercice
        WHERE motscles LIKE :pattern
        ';
        $stmt = $conn->prepare($sql);
        $stmt->execute(['pattern' => $cle]);
        return $stmt->fetchAll(); // returne des arrays, pas des objects
    }


}
```

Mon repositoire

. . . et avec DQL (Doctrine Query Language) :

```php
<?php

// src/AppBundle/Repository/ExerciceRepository.php
namespace AppBundle\Repository;

use Doctrine\ORM\EntityRepository;

class ExerciceRepository extends EntityRepository
{

    public function findWithMotCleDQL($cle){
        $entityManager = $this->getEntityManager();
        $query = $entityManager->createQuery(
          'SELECT p
          FROM AppBundle\Entity\Exercice p
          WHERE p.motscles LIKE :cle
           ')->setParameter('cle',"%$cle%");
        $sql=$query->getSQL();
        return $query->execute();
    }
}
```

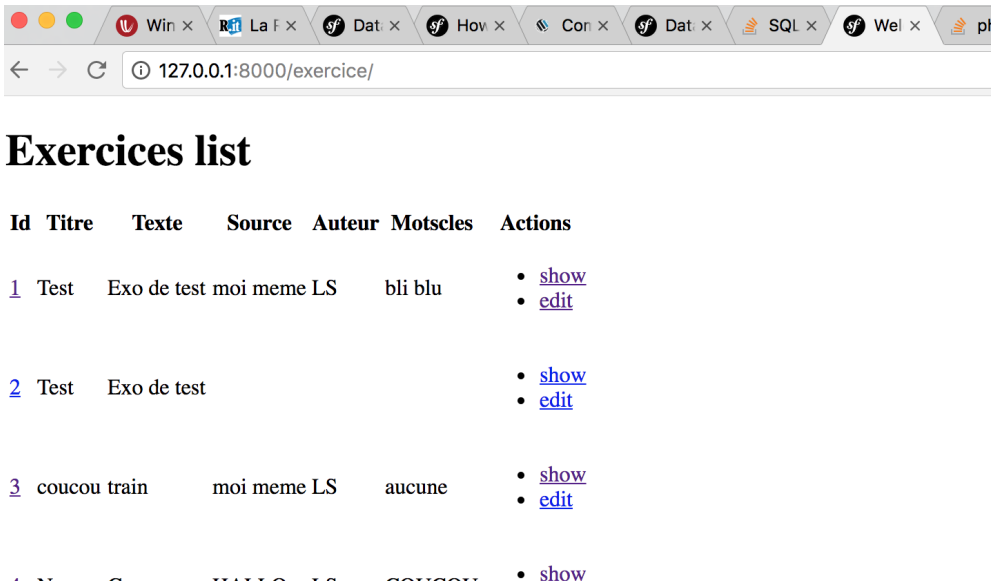
Plan

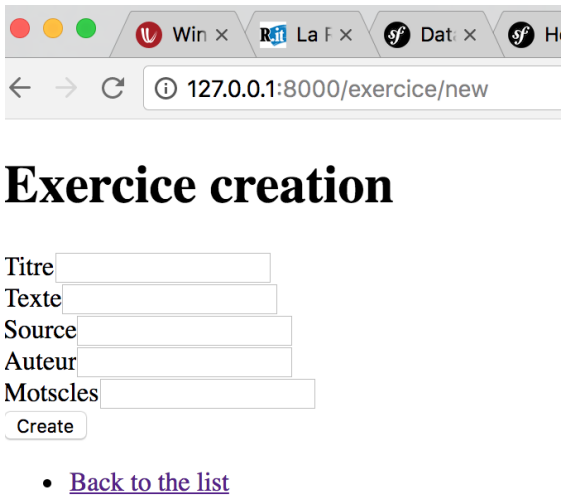
Génération d'une interface pour administrer les entités

CRUD : Create, Read, Update, Delete(Créer, Lire, Mettre à jour, Supprimer).
La commande

```
php bin/console doctrine:generate:crud
```

permet d'engendrer un formulaire et un contrôleur pour administrer les entités.





Le contrôleur *ExerciceController.php*

```php
<?php

namespace AppBundle\Controller;

use AppBundle\Entity\Exercice;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;use
 ↪   Symfony\Component\HttpFoundation\Request;

/**
 * Exercice controller.
 *
 * @Route("exercice")
 */
class ExerciceController extends Controller
{
```

Le contrôleur *ExerciceController.php*

```
17      /**
18       * Lists all exercice entities.
19       *
20       * @Route("/", name="exercice_index")
21       * @Method("GET")
22       */
23      public function indexAction()
24      {
25          $em = $this->getDoctrine()->getManager();
26
27          $exercices = $em->getRepository('AppBundle:Exercice')->findAll();
28
29          return $this->render('exercice/index.html.twig', array(
30              'exercices' => $exercices,
31          ));
32      }
```

Le contrôleur *ExerciceController.php*

```
34      /**
35       * Creates a new exercice entity.
36       *
37       * @Route("/new", name="exercice_new")
38       * @Method({"GET", "POST"})
39       */
40      public function newAction(Request $request)
41      {
42          $exercice = new Exercice();
43          $form = $this->createForm('AppBundle\Form\ExerciceType', $exercice);
44          $form->handleRequest($request);
45
46          if ($form->isSubmitted() && $form->isValid()) {
47              $em = $this->getDoctrine()->getManager();
48              $em->persist($exercice);
49              $em->flush();
50
51              return $this->redirectToRoute('exercice_show', array('id' =>
    ↪   $exercice->getId()));
52          }
53
54          return $this->render('exercice/new.html.twig', array(
55              'exercice' => $exercice,
56              'form' => $form->createView(),
57          ));
58      }
```

Le contrôleur *ExerciceController.php*

```
60      /**
61       * Finds and displays a exercice entity.
62       *
63       * @Route("/{id}", name="exercice_show")
64       * @Method("GET")
65       */
66      public function showAction(Exercice $exercice)
67      {
68          $deleteForm = $this->createDeleteForm($exercice);
69
70          return $this->render('exercice/show.html.twig', array(
71              'exercice' => $exercice,
72              'delete_form' => $deleteForm->createView(),
73          ));
74      }
```

Le contrôleur *ExerciceController.php*

```
76      /**
77       * Displays a form to edit an existing exercice entity.
78       *
79       * @Route("/{id}/edit", name="exercice_edit")
80       * @Method({"GET", "POST"})
81       */
82      public function editAction(Request $request, Exercice $exercice)
83      {
84          $deleteForm = $this->createDeleteForm($exercice);
85          $editForm = $this->createForm('AppBundle\Form\ExerciceType', $exercice);
86          $editForm->handleRequest($request);
87
88          if ($editForm->isSubmitted() && $editForm->isValid()) {
89              $this->getDoctrine()->getManager()->flush();
90
91              return $this->redirectToRoute('exercice_edit', array('id' =>
    ↪   $exercice->getId()));
92          }
93
94          return $this->render('exercice/edit.html.twig', array(
95              'exercice' => $exercice,
96              'edit_form' => $editForm->createView(),
97              'delete_form' => $deleteForm->createView(),
98          ));
99      }
```

Le contrôleur *ExerciceController.php*

```
101     /**
102      * Deletes a exercice entity.
103      *
104      * @Route("/{id}", name="exercice_delete")
105      * @Method("DELETE")
106      */
107     public function deleteAction(Request $request, Exercice $exercice)
108     {
109         $form = $this->createDeleteForm($exercice);
110         $form->handleRequest($request);
111 
112         if ($form->isSubmitted() && $form->isValid()) {
113             $em = $this->getDoctrine()->getManager();
114             $em->remove($exercice);
115             $em->flush();
116         }
117 
118         return $this->redirectToRoute('exercice_index');
119     }
```

Le contrôleur *ExerciceController.php*

```
121     /**
122      * Creates a form to delete a exercice entity.
123      *
124      * @param Exercice $exercice The exercice entity
125      *
126      * @return \Symfony\Component\Form\Form The form
127      */
128     private function createDeleteForm(Exercice $exercice)
129     {
130         return $this->createFormBuilder()
131             ->setAction($this->generateUrl('exercice_delete', array('id' =>
    ↪    $exercice->getId())))
132             ->setMethod('DELETE')
133             ->getForm()
134         ;
135     }
136 }
```

Une preview des forms

```php
<?php

namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ExerciceType extends AbstractType
{

    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('titre')->add('texte')->add('source')->add('auteur')->add('motscles');
    }/**


}
```

Une preview des forms

```
9   class ExerciceType extends AbstractType
10  {

18      * {@inheritdoc}
19      */
20      public function configureOptions(OptionsResolver $resolver)
21      {
22          $resolver->setDefaults(array(
23              'data_class' => 'AppBundle\Entity\Exercice'
24          ));
25      }
26
27      /**
28       * {@inheritdoc}
29       */
30      public function getBlockPrefix()
31      {
32          return 'appbundle_exercice';
33      }

36  }
```

Une preview de twig

```
1  {% extends 'base.html.twig' %}
2
3  {% block body %}
4      <h1>Exercice edit</h1>
5
6      {{ form_start(edit_form) }}
7          {{ form_widget(edit_form) }}
8          <input type="submit" value="Edit" />
9      {{ form_end(edit_form) }}
10
11     <ul>
12         <li>
13             <a href="{{ path('exercice_index') }}">Back to the list</a>
14         </li>
15         <li>
16             {{ form_start(delete_form) }}
17                 <input type="submit" value="Delete">
18             {{ form_end(delete_form) }}
19         </li>
20     </ul>
21 {% endblock %}
```