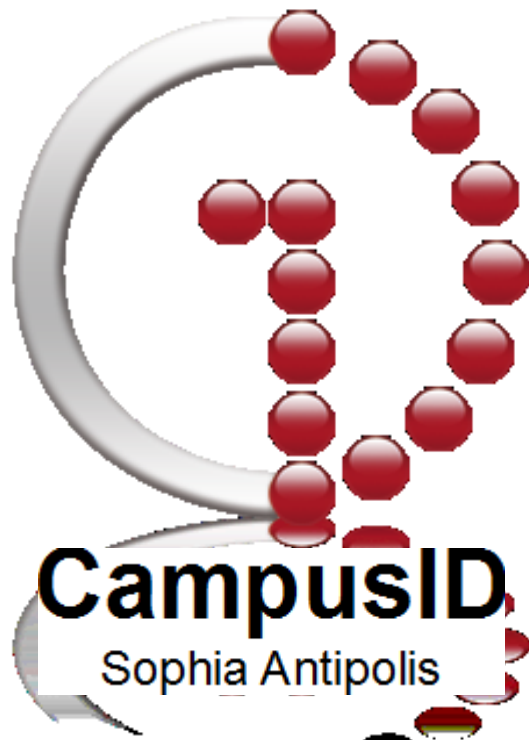


Software PHP

Framework Symfony 3

PARTIE 3 – Gérer la base de données avec Doctrine2

B2/B3



Version 1.1

Crédits et références bibliographiques

- [Site web de Symfony](#)
- [Site web de Sensiolabs](#)
- [Livre de référence : Développez votre site web avec le framework Symfony3](#)

- PARTIE 3 – Gérer la base de données avec Doctrine2
 - La couche métier : les entités
 - Manipuler ses entités avec Doctrine2
 - Gérer des formulaires

PARTIE 3 - Gérer la base de données avec Doctrine2

La couche métier : les entités

Notions d'ORM

- L'objectif d'un **ORM** (Object-Relation Mapper) est d'**abstraire une base de données** en se chargeant de l'enregistrement de vos données
- On n'aura plus à écrire de requêtes, ni créer de tables via phpMyAdmin.
- L'ORM par défaut de Symfony s'appelle **Doctrine2**
- Toutes les **données** doivent être **sous forme d'objets** que l'on nomme **entité** (*entity*)
- On va indiquer à Symfony ce qu'il doit faire à travers des **commentaires spéciaux** dans le code (*metadata*)

Exemple d'annotation

- Voici à quoi pourrait ressembler la classe Advert avant/après l'annotation :

```
<?php
// src/OC/PlatformBundle/Entity/Advert.php

namespace OC\PlatformBundle\Entity;

class Advert
{
    protected $id;
    protected $content;

    // Et bien sûr les getters/setters :

    public function setId($id)
    {
        $this->id = $id;
    }

    ..../....
```

```
.../....

use Doctrine\ORM\Mapping as ORM;

/** ← Notez le /**
 * @ORM\Entity
 */
class Advert
{
    /**
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(name="date", type="date")
     */
    protected $date;

    .../...
```

Création de la base de données

- Avant d'utiliser le générateur d'entité il faut veiller à :
 1. Modifier le fichier `app/config/parameters.yml` pour adapter les paramètres utiles à la base de données (utilisateur, mot de passe, nom de la base de données...)

```
parameters:
    database_host: 127.0.0.1
    database_port: null
    database_name: symfony
    database_user: root
    database_password: null
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
```

2. Ensuite on peut lancer la commande de création de la base de données :

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

```
C:\wamp\www\Symfony3>php bin/console doctrine:database:create
Created database `symfony` for connection named default
```


Génération d'entité

- Une fois la base de données créée, on peut créer une entité

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

- Première étape : le nom, sous le **format NomBundle:NomEntité**. Dans notre cas, on entre donc `OCPlatformBundle:Advert`

```
C:\wamp\www\Symfony3>php bin/console doctrine:generate:entity
```

```
Welcome to the Doctrine2 entity generator
```

```
This command helps you generate Doctrine2 entities.
```

```
First, you need to give the entity name you want to generate.  
You must use the shortcut notation like AcmeBlogBundle:Post.
```

```
The Entity shortcut name: OCPlatformBundle:Advert
```

Génération d'entité – configuration

- La seconde étape consiste à choisir une configuration, dans notre cas on utilisera les annotations (option par défaut)
- On valide juste par entrée

```
The Entity shortcut name: OCPlatformBundle:Advert
```

```
Determine the format to use for the mapping information.
```

```
Configuration format (yaml, xml, php, or annotation) [annotation]:
```

Génération d'entité – création de champs

- On peut créer des champs pour notre base de données
- Différentes types sont disponibles
- Créons le premier champ `date` de type `datetime`
- Ce champ n'est pas facultatif (*is nullable*), ni unique (choix par défaut)

```
Instead of starting with a blank entity, you can add some fields now.  
Note that the primary key will be added automatically (named id).
```

```
Available types: array, simple_array, json_array, object,  
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,  
date, time, decimal, float, binary, blob, guid.
```

```
New field name (press <return> to stop adding fields): date  
Field type [string]: datetime  
Is nullable [false]:  
Unique [false]:
```

Génération d'entité – création de champs

- De la même façon, voici les autres champs utiles pour notre exemple :

```
New field name (press <return> to stop adding fields): title
Field type [string]:
Field length [255]:
Is nullable [false]:
Unique [false]:
```

```
New field name (press <return> to stop adding fields): author
Field type [string]:
Field length [255]:
Is nullable [false]:
Unique [false]:
```

```
New field name (press <return> to stop adding fields): content
Field type [string]: text
Is nullable [false]:
Unique [false]:
```

Génération d'entité – création de champs

- Après le dernier champ valider par entrée

```
New field name (press <return> to stop adding fields):
```

```
Entity generation
```

```
created .\src\OC\PlatformBundle\Entity/
```

```
created .\src\OC\PlatformBundle\Entity\Advert.php
```

```
> Generating entity class C:\wamp\www\Symfony3\src\OC\PlatformBundle\Entity\Advert.php: OK!
```

```
> Generating repository class C:\wamp\www\Symfony3\src\OC\PlatformBundle\Repository\AdvertRepository.php: OK!
```

```
Everything is OK! Now get to work :).
```

Fichier Entity/Advert.php

- On peut vérifier que Symfony a généré toutes les annotations nécessaires

```

<?php
namespace OC\PlatformBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Advert
 *
 * @ORM\Table(name="advert")
 * @ORM\Entity(repositoryClass="OC\PlatformBundle\Repository\AdvertRepository")
 */
class Advert
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
  
```

Table (permet de personnaliser le nom de la table qui sera créée dans la base de données)

Déclaration d'entité

Colonne (permet de définir les caractéristiques de la colonne concernée)

Le fait d'annoter les objets s'appelle le **mapping**

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/basic-mapping.html>

Ajout de méthodes à classe entité

- On peut si besoin ajouter dans la classe de l'entité des méthodes qui utilisent les attributs issus de la base données pour retourner des valeurs (exemple : une méthode `getPrixTotal` qui ferait la somme des prix des produits d'une commande)

```
<?php
// Exemple :
class Commande
{
    public function getPrixTotal()
    {
        $prix = 0;
        foreach($this->getListeProduits() as $produit) {
            $prix += $produit->getPrix();
        }
        return $prix;
    }
}
```

Attributs par défaut

- On peut grâce au constructeur définir des valeurs par défaut pour certains des attributs de l'entité

```
/**
 * Advert
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="OC\PlatformBundle\Entity\AdvertRepository")
 */
class Advert
{
    // ...

    public function __construct()
    {
        // Par défaut, la date de l'annonce est la date d'aujourd'hui
        $this->date = new \DateTime();
    }

    // ...
}
```


Les types de colonnes

- Ils ressemblent à ceux de SQL et PHP mais il ne s'agit que d'une correspondance. Ce sont donc des types de Doctrine uniquement.

| Type Doctrine | Type SQL | Type PHP | Utilisation |
|------------------|------------------------------------|------------------------|---|
| string | VARCHAR | string | Toutes les chaînes de caractères jusqu'à 255 caractères. |
| integer | INT | integer | Tous les nombres jusqu'à 2 147 483 647. |
| smallint | SMALLINT | integer | Tous les nombres jusqu'à 32 767. |
| bigint | BIGINT | string | Tous les nombres jusqu'à 9 223 372 036 854 775 807. Attention, PHP reçoit une chaîne de caractères, car il ne supporte pas un si grand nombre (suivant que vous êtes en 32 ou en 64 bits). |
| boolean | BOOLEAN(Oracle)/TINYINT(1) (MySQL) | boolean | Les valeurs booléennes true et false. |
| decimal | DECIMAL | double | Les nombres à virgule. |
| date ou datetime | DATETIME | objet DateTime | Toutes les dates et heures. |
| time | TIME | objet DateTime- | Toutes les heures. |
| text | CLOB(Oracle)/TEXT(MySQL) | string | Les chaînes de caractères de plus de 255 caractères. |
| object | CLOB/TEXT | Type de l'objet stocké | Stocke un objet PHP en utilisant serialize/unserialize. |
| array | CLOB/TEXT | array | Stocke un tableau PHP en utilisant serialize/unserialize. |
| float | FLOAT | double | Tous les nombres à virgule. Attention, fonctionne uniquement sur les serveurs dont la locale utilise un point comme séparateur. |

Les paramètres de l'annotation `Column`

- Il existe 7 paramètres, tous facultatifs, que l'on peut passer à l'annotation `Column` afin de personnaliser le comportement.

| Paramètre | Valeur par défaut | Utilisation |
|------------------------|---------------------|--|
| <code>type</code> | <code>string</code> | Définit le type de colonne comme nous venons de le voir. |
| <code>name</code> | Nom de l'attribut | Définit le nom de la colonne dans la table. Par défaut, le nom de la colonne est le nom de l'attribut de l'objet, ce qui convient parfaitement. Mais vous pouvez changer le nom de la colonne, par exemple si vous préférez « <code>isExpired</code> » en attribut, mais « <code>is_expired</code> » dans la table. |
| <code>length</code> | 255 | Définit la longueur de la colonne. Applicable uniquement sur un type de colonne <code>string</code> . |
| <code>unique</code> | <code>false</code> | Définit la colonne comme unique. Par exemple sur une colonne e-mail pour vos membres. |
| <code>nullable</code> | <code>false</code> | Permet à la colonne de contenir des <code>NULL</code> . |
| <code>precision</code> | 0 | Définit la précision d'un nombre à virgule, c'est-à-dire le nombre de chiffres en tout. Applicable uniquement sur un type de colonne <code>decimal</code> . |
| <code>scale</code> | 0 | Définit le <i>scale</i> d'un nombre à virgule, c'est-à-dire le nombre de chiffres après la virgule. Applicable uniquement sur un type de colonne <code>decimal</code> . |

- Le rôle d'un **ORM** est d'**abstraire une base de données** en se chargeant de l'enregistrement de vos données
- L'ORM par défaut livré avec Symfony est **Doctrine2**.
- L'utilisation d'un ORM implique l'utilisation **d'objets**.
- 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 correctement l'objet en base de données.

Manipuler ses entités avec Doctrine2

Matérialiser les tables en base de données

- L'objectif de ce chapitre est de voir comment on manipule des entités à l'aide de Doctrine2.
- La première chose à faire est de créer la requête pour la génération de tables dans la base de données :

```
php bin/console doctrine:schema:update  
--dump-sql
```

```
C:\wamp\www\Symfony3>php bin/console doctrine:schema:update --dump-sql  
CREATE TABLE advert (id INT AUTO_INCREMENT NOT NULL, date DATETIME NOT NULL,  
title VARCHAR(255) NOT NULL, author VARCHAR(255) NOT NULL, content LONGTEXT N  
OT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci  
ENGINE = InnoDB;
```

- A ce stade les tables ne sont pas encore créées



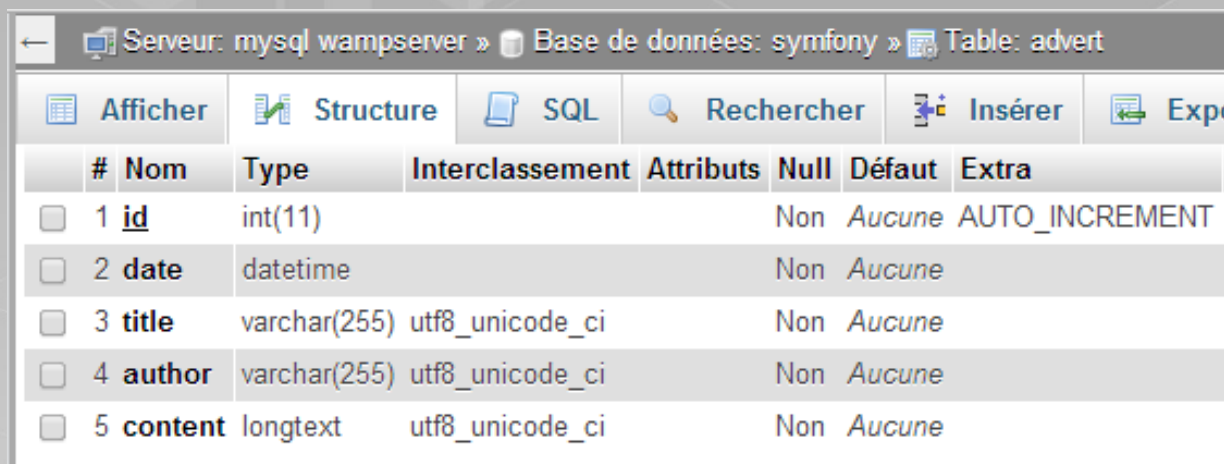
Matérialiser les tables en base de données

- Une fois que vous avez validé la requête, vous pouvez l'exécuter :

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

```
C:\wamp\www\Symfony3>php bin/console doctrine:schema:update --force
Updating database schema...
Database schema updated successfully! "1" query was executed
```

- Et aller vérifier dans PHPMysqlAdmin que la table `advert` a été créée :



The screenshot shows the PHPMysqlAdmin interface for a MySQL database named 'symfony'. The selected table is 'advert'. The 'Structure' tab is active, displaying the table's schema. The table has five columns: 'id' (int(11), primary key, auto-increment), 'date' (datetime), 'title' (varchar(255), utf8_unicode_ci), 'author' (varchar(255), utf8_unicode_ci), and 'content' (longtext, utf8_unicode_ci). All columns are non-null and have no default values.

| # | Nom | Type | Interclassement | Attributs | Null | Défaut | Extra |
|----------------------------|-----------|------------------------------|-----------------|-----------|------|--------|----------------|
| <input type="checkbox"/> 1 | <u>id</u> | int(11) | | | Non | Aucune | AUTO_INCREMENT |
| <input type="checkbox"/> 2 | date | datetime | | | Non | Aucune | |
| <input type="checkbox"/> 3 | title | varchar(255) utf8_unicode_ci | | | Non | Aucune | |
| <input type="checkbox"/> 4 | author | varchar(255) utf8_unicode_ci | | | Non | Aucune | |
| <input type="checkbox"/> 5 | content | longtext utf8_unicode_ci | | | Non | Aucune | |

Modifier une entité

- Pour modifier une entité, il suffit de lui créer un attribut et de lui attacher l'annotation correspondante.
- Par exemple ajoutons un attribut `$published`, un booléen qui indique si l'annonce est publiée (true pour l'afficher sur la page d'accueil, false sinon)

```
/**
 * Advert
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="OC\PlatformBundle\Entity\AdvertRepository")
 */
class Advert
{
    // ... les autres attributs

    /**
     * @ORM\Column(name="published", type="boolean")
     */
    private $published = true;

    // ...
}
```

Mise à jour de la classe

- Vous pouvez créer manuellement les setters et getters ou bien les faire générer par Symfony

```
php bin/console doctrine:generate:entities  
OCPlatformBundle:Advert
```

```
C:\wamp\www\Symfony3>php bin/console doctrine:generate:entities OCPlatformBundle:Advert  
Generating entity "OCPlatformBundle\Entity\Advert"  
> backing up Advert.php to Advert.php~  
> generating OCPlatformBundle\Entity\Advert
```

```
/**  
 * Set published  
 *  
 * @param boolean $published  
 *  
 * @return Advert  
 */  
public function setPublished($published)  
{  
    $this->published = $published;  
  
    return $this;  
}  
  
/**  
 * Get published  
 *  
 * @return boolean  
 */  
public function getPublished()  
{  
    return $this->published;  
}
```


Mise à jour de la base de données

- Vient ensuite la mise à jour de la base de données
 - Tout d'abord la création de la requête via la commande :

```
php bin/console doctrine:schema:update --dump-sql
```

```
C:\wamp\www\Symfony3>php bin/console doctrine:schema:update --dump-sql
ALTER TABLE advert ADD published TINYINT(1) NOT NULL;
```

- Puis son exécution via la commande :

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

```
C:\wamp\www\Symfony3>php bin/console doctrine:schema:update --force
Updating database schema...
Database schema updated successfully! "1" query was executed
```

☐ 6 published tinyint(1)

Non Aucune

 Modifier  Supprimer

Les requêtes SQL avec le service EntityManager

- Pour faire des requête SQL on va utiliser le service EntityManager
- On accède à ce service via :

```
$em = $this->getDoctrine()->getManager();
```

- Pour récupérer les entités de la base de données, on va utiliser des objets Repository (un par entité) :

```
$em = $this->getDoctrine()->getManager();
```

```
$advertRepository = $em->getRepository('OCPlatformBundle:Advert');
```

- Note : on peut aussi utiliser l'espace de nom complet :

```
getRepository('OC\PlatformBundle\Entity\Advert');
```

Enregistrement en base de données

- L'enregistrement effectif en base de données se fait en deux étapes très simples depuis un contrôleur.
- Modifions la méthode `addAction()` de notre contrôleur

```
<?php
// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use OC\PlatformBundle\Entity\Advert;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class AdvertController extends Controller
{
    public function addAction(Request $request)
    {
        // Création de l'entité
        $advert = new Advert();
        $advert->setTitle('Recherche développeur Symfony.');
```

```
        // On récupère l'EntityManager
        $em = $this->getDoctrine()->getManager();

        // Étape 1 : On « persiste » l'entité
        $em->persist($advert);

        // Étape 2 : On « flush » tout ce qui a été persisté avant
        $em->flush();

        // Reste de la méthode qu'on avait déjà écrit
        if ($request->isMethod('POST')) {
            $request->getSession()->getFlashBag()->add('notice', 'Annonce
            bien enregistrée.');
```

Enregistrer ses entités en base de données

- L'étape 1 dit à Doctrine de « persister » l'entité. Cela veut dire qu'à partir de maintenant cette nouvelle entité est gérée par Doctrine. Cela n'exécute pas encore de requête SQL, ni rien d'autre.
- L'étape 2 dit à Doctrine d'exécuter effectivement les requêtes nécessaires pour sauvegarder les entités qu'on lui a dit de persister précédemment (il fait donc des INSERT INTO ...)
- On a ainsi ajouté une annonce dans la base de données :

http://localhost/Symfony/web/app_dev.php/platform/add

+ Options

| | id | date | title | author | content | published |
|--|----|---------------------|--------------------------------|-----------|---|-----------|
| <input type="checkbox"/> Modifier <input type="checkbox"/> Copier <input type="checkbox"/> Effacer | 1 | 2017-02-10 15:33:20 | Recherche développeur Symfony. | Alexandre | Nous recherchons un développeur Symfony débutant s... | 1 |

Visualisation des requêtes avec le profiler

- L'outil profiler permet de visualiser les requêtes SQL :

Database Queries3

Query time6.00 ms

Invalid entities0

Second Level Cachedisabled

200 @ oc_platform_add 501 ms 4.0 MB anon. 42 ms 3 in 6.00 ms

Queries

Group similar statements

| #▲ | Time | Info |
|----|---------|---|
| 1 | 0.56 ms | "START TRANSACTION" Parameters: [] View formatted query View runnable query Explain query |
| 2 | 1.74 ms | INSERT INTO advert (date, title, author, content, published) VALUES (?, ?, ?, ?, ?) Parameters: [1 => 2017-02-10 15:33:20, 2 => Recherche développeur Symfony., 3 => Alexandre, 4 => Nous recherchons un développeur Symfony débutant sur Lyon. Blabla..., 5 => 1] View formatted query View runnable query Explain query |
| 3 | 3.93 ms | "COMMIT" Parameters: [] View formatted query View runnable query Explain query |

Remarques sur persist et flush

- Les étapes `persist` et `flush` sont séparées car Doctrine utilise les transactions : évite ainsi des écritures partielles
- `persist` traite indifféremment les nouvelles entités de celles déjà en base de données

```
<?php
// Depuis un contrôleur

$em = $this->getDoctrine()->getManager();

// On crée une nouvelle annonce
$advert1 = new Advert;
$advert1->setTitle('Recherche développeur.');
```

`$advert1->setContent("Pour mission courte");`

```
// Et on le persiste
$em->persist($advert1);

// On récupère l'annonce d'id 5.
$advert2 = $em->getRepository('OCPlatformBundle:Advert')->find(5);

// On modifie cette annonce, en changeant la date à la date d'aujourd'hui
$advert2->setDate(new \Datetime());

// Ici, pas besoin de faire un persist() sur $advert2.

// Enfin, on applique les deux changements à la base de données :
$em->flush();
```

Autres méthodes utiles de l'EntityManager

- `clear($nomEntite)` annule tous les `persist()` effectués. Si le nom d'une entité est précisé (son namespace complet ou son raccourci), seuls les `persist()` sur des entités de ce type seront annulés.
- `detach($entite)` annule le `persist()` effectué sur l'entité en argument. Au prochain `flush()`, aucun changement ne sera donc appliqué à l'entité.
- `contains($entite)` retourne `true` si l'entité donnée en argument est gérée par l'EntityManager (s'il y a déjà eu un `persist()` sur cette entité).
- `refresh($entite)` met à jour l'entité donnée en argument dans l'état où elle est en base de données. Cela écrase et donc annule tous les changements qu'il a pu y avoir sur l'entité concernée.
- `remove($entite)` supprime l'entité donnée en argument de la base de données. Effectif au prochain `flush()`

Suppression d'annonce

- On modifie la méthode `deleteAction()` :

```
public function deleteAction(Request $request, $id)
{
    $em = $this->getDoctrine()->getManager();
    $advert = $em->getRepository('OCPlatformBundle:Advert')->find($id);
    if (null === $advert) {
        throw new NotFoundException("L'annonce d'id ".$id." n'existe pas.");
    }

    $em->remove($advert);
    $em->flush();
    $request->getSession()->getFlashBag()->add('info', "L'annonce a bien été supprimée.");
    return $this->redirectToRoute('oc_platform_home');
}
```


Récupérer ses entités avec un EntityRepository

- Depuis un repository, il faut utiliser la méthode `find($id)` qui permet de retourner l'entité correspondant à l'id `$id`

```
public function viewAction($id)
{
    // On récupère le repository
    $repository = $this->getDoctrine()->getManager()->getRepository('OCPlatformBundle:Advert');

    // On récupère l'entité correspondante à l'id $id
    $advert = $repository->find($id);

    // $advert est donc une instance de OC\PlatformBundle\Entity\Advert
    // ou null si l'id $id n'existe pas, d'où ce if :
    if (null === $advert) {
        throw new NotFoundHttpException("L'annonce d'id ".$id." n'existe pas.");
    }

    // Le render ne change pas, on passait avant un tableau, maintenant un objet
    return $this->render('OCPlatformBundle:Advert:view.html.twig', array(
        'advert' => $advert
    ));
}
```

Récupérer ses entités avec un EntityRepository

http://localhost/Symfony/web/app_dev.php/platform/advert/1

Ma plateforme d'annonces

Ce projet est propulsé par Symfony.

Ecole supérieure d'informatique»

Les annonces

Accueil

Ajouter une annonce

Dernières annonces

Recherche développeur Symfony

Mission de webmaster

Offre de stage webdesigner

Annonces

Recherche développeur Symfony.

Par Alexandre, le 09/11/2017

Nous recherchons un développeur Symfony débutant sur Lyon. Blabla...


Retour à la liste

Modifier l'annonce

Supprimer l'annonce

The sky's the limit © 2017 and beyond.

40



CampusID
Sophia Antipolis

Autres méthodes utiles de récupération – findAll()

- `findAll()` : retourne toutes les entités contenue dans la base de données. Le format du retour est un tableau PHP

```
<?php
$repository = $this
->getDoctrine()
->getManager()
->getRepository('OCPlatformBundle:Advert')
;

$listAdverts = $repository->findAll();
```

Affichage en PHP

```
foreach ($listAdverts as $advert) {
    // $advert est une instance de Advert
    echo $advert->getContent();
}
```

Affichage en twig
(si l'on a passé la
variable `$listAdverts`
au template)

```
<ul>
    {% for advert in listAdverts %}
        <li>{{ advert.content }}</li>
    {% endfor %}
</ul>
```

Autres méthodes utiles de récupération – findBy()

- `findBy()` : permet de retourner une liste d'entités, sauf qu'elle est capable d'effectuer un filtre pour ne retourner que les entités correspondant à un ou plusieurs critère(s)

```
<?php

$listAdverts = $repository->findBy(
    array('author' => 'Alexandre'), // Critere
    array('date' => 'desc'),        // Tri
    5,                               // Limite
    0                                // Offset
);
```

Cet exemple va récupérer toutes les entités ayant comme auteur « Alexandre » en les classant par date décroissante et en en sélectionnant cinq(5) à partir du début(0). Elle retourne un tableau également

1^{ère} Application : Affichage des annonces sur l'index

- Pour cela il faut utiliser la méthode `findAll()` et on modifie la méthode `indexAction` comme suit :

```
public function indexAction($page)
{
    if ($page < 1) {
        throw new NotFoundException('Page "'.$page.'" inexistante.');
```

```
    }

    // Pour récupérer la liste de toutes les annonces : on utilise findAll()
    $listAdverts = $this->getDoctrine()
        ->getManager()
        ->getRepository('OCPlatformBundle:Advert')
        ->findAll()
    ;

    // L'appel de la vue ne change pas
    return $this->render('OCPlatformBundle:Advert:index.html.twig', array(
        'listAdverts' => $listAdverts,
    ));
}
```

2^{ème} Application : Affichage des annonces dans le menu

- On modifie également la méthode `menuAction` par exemple en utilisant la méthode `findBy()` :

```
public function menuAction($limit)
{
    $em = $this->getDoctrine()->getManager();

    $listAdverts = $em->getRepository('OCPlatformBundle:Advert')->findBy(
        array(),           // Pas de critère
        array('date' => 'desc'), // On trie par date décroissante
        $limit,           // On sélectionne $limit annonces
        0                 // À partir du premier
    );

    return $this->render('OCPlatformBundle:Advert:menu.html.twig', array(
        'listAdverts' => $listAdverts
    ));
}
```

Affichage de la page d'accueil

http://localhost/symfony/web/app_dev.php/platform

Ma plateforme d'annonces

Ce projet est propulsé par Symfony.

Ecole supérieure d'informatique»

Les annonces

[Accueil](#)

[Ajouter une annonce](#)

Dernières annonces

[Webmaster](#)

[Recherche développeur Symfony.](#)

Annonces

Liste des annonces

- [Recherche développeur Symfony.](#) par Alexandre, le 09/11/2017
- [Webmaster](#) par LB, le 14/11/2017

The sky's the limit © 2017 and beyond.

- Il faut exécuter la commande `doctrine:schema:update` pour mettre à jour la base de données et la faire correspondre à l'état actuel de vos entités.
- Avec Symfony, on récupère l'EntityManager de Doctrine2 depuis un contrôleur, via `$this->getDoctrine()->getManager()`.
- L'EntityManager sert à manipuler les entités, tandis que les *repositories* servent à récupérer les entités.

Gérer des formulaires

Création de formulaire

- Les formulaires représentent souvent l'interface avec les utilisateurs d'un site
- La création de formulaire va de paire avec la validation (sécurisation) des données
- Dans Symfony, un formulaire se construit sur un objet existant, et son objectif est d'hydrater cet objet
- Pour créer des formulaires, Symfony propose le composant Form

Objectif : hydrater l'objet

- Prenons l'exemple d'AdvertController, on commence par ajouter les différents espaces de nom

```
<?php
// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use OC\PlatformBundle\Entity\Advert;

use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\FormType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
```

Objectif : hydrater l'objet

- Puis on utilise le service form factory :

```
public function addAction(Request $request)
{
    // On crée un objet Advert
    $advert = new Advert();

    // On crée le FormBuilder grâce au service form factory
    $formBuilder = $this->get('form.factory')->createBuilder(FormType::class,
    $advert);

    // On ajoute les champs de l'entité que l'on veut à notre formulaire
    $formBuilder->add('date',      DateType::class);
    $formBuilder->add('title',     TextType::class);
    $formBuilder->add('content',   TextareaType::class);
    $formBuilder->add('author',    TextType::class);
    $formBuilder->add('published', CheckboxType::class);
    $formBuilder->add('save',      SubmitType::class);
}
```

Note : `DateType::class` est un raccourci pour :

`'Symfony\Component\Form\Extension\Core\Type\TextType'`

Objectif : hydrater l'objet

- On termine en ajoutant la création du formulaire et le passage à la vue

```
// À partir du FormBuilder, on génère le formulaire
$form = $formBuilder->getForm();

// On passe la méthode createView() du formulaire à la vue
// afin qu'elle puisse afficher le formulaire toute seule
return $this->render('OCPlatformBundle:Advert:add.html.twig', array(
    'form' => $form->createView(),
));
```

- Modifiez la vue Advert/form.html.twig comme suit :

```
{# src/OC/PlatformBundle/Resources/views/Advert/form.html.twig #}

<h3>Formulaire d'annonce</h3>

<div class="well">
    {{ form(form) }}
</div>
```

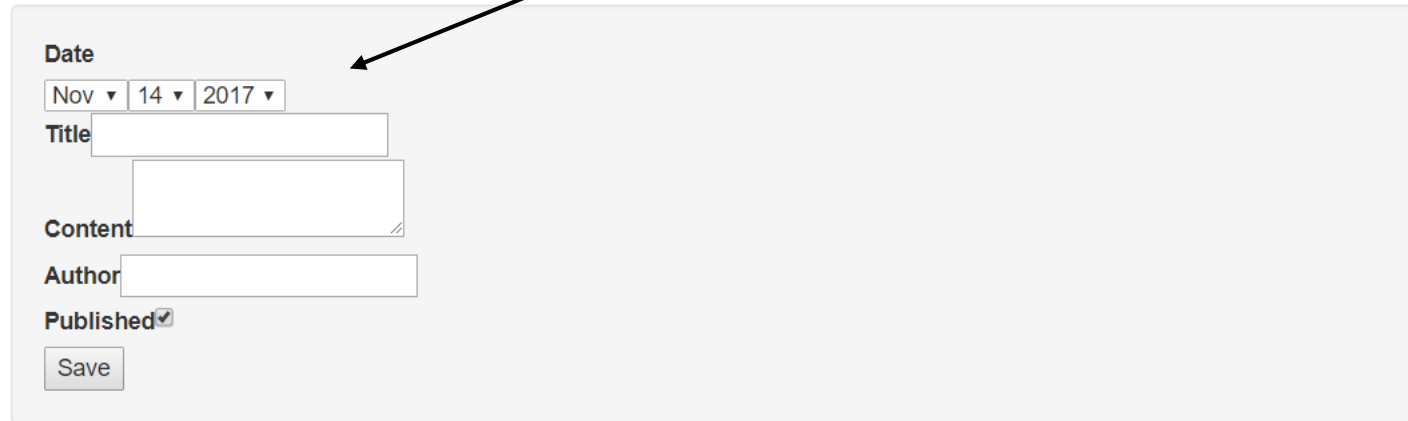
Objectif : hydrater l'objet

- Le visuel :

http://localhost/symfony/web/app_dev.php/platform/add

Ajouter une annonce

Formulaire d'annonce



Date

Nov ▼ 14 ▼ 2017 ▼

Title

Content

Author

Published ☒

Save

Attention : cette annonce sera ajoutée directement sur la page d'accueil après validation du formulaire.

Les types de champs

- Vous pouvez choisir parmi ces types :

| Texte | Choix | Date et temps | Divers | Multiple | Caché |
|--------------|---------------------|---------------|--------------|----------------|------------|
| TextType | ChoiceType | DateType | CheckboxType | CollectionType | HiddenType |
| TextareaType | EntityType | DatetimeType | FileType | RepeatedType | CsrfType |
| EmailType | CountryType | TimeType | RadioType | | |
| IntegerType | LanguageType | BirthdayType | | | |
| MoneyType | LocaleType | | | | |
| NumberType | TimezoneType | | | | |
| PasswordType | <u>CurrencyType</u> | | | | |
| PercentType | | | | | |
| SearchType | | | | | |
| UrlType | | | | | |
| RangeType | | | | | |

Gestion de la soumission d'un formulaire

- Il faut tout d'abord vérifier que la requête est de type POST
- Il faut ensuite faire le lien entre les variables de type POST et le formulaire. Pour cela on utilise la méthode `handleRequest()` qui gère à la fois la récupération des données et l'hydratation.
- Enfin il faudra vérifier que les données sont valides c'est-à-dire cohérentes avec ce que l'objet et le formulaire attendent. Cela se fait via la méthode `isValid()` du formulaire.

Gestion de la soumission d'un formulaire

- Il faut tout d'abord vérifier que la requête est de type POST
- Il faut ensuite faire le lien entre les variables de type POST et le formulaire. Pour cela on utilise la méthode `handleRequest()` qui gère à la fois la récupération des données et l'hydratation.
- Enfin il faudra vérifier que les données sont valides c'est-à-dire cohérentes avec ce que l'objet et le formulaire attendent. Cela se fait via la méthode `isValid()` du formulaire.

Gestion de la soumission d'un formulaire

- Voici le code de la méthode `addAction()` :

```
public function addAction(Request $request)
{
    // On crée un objet Advert
    $advert = new Advert();

    // On crée le FormBuilder grâce au service form factory
    $formBuilder = $this->get('form.factory')->createBuilder(FormType::class,
    $advert);

    // On ajoute les champs de l'entité que l'on veut à notre formulaire
    $formBuilder->add('date',      DateType::class);
    $formBuilder->add('title',     TextType::class);
    $formBuilder->add('content',   TextareaType::class);
    $formBuilder->add('author',    TextType::class);
    $formBuilder->add('published', CheckboxType::class);
    $formBuilder->add('save',      SubmitType::class);

    // À partir du FormBuilder, on génère le formulaire
    $form = $formBuilder->getForm();

    // ...
}
```

Gestion de la soumission d'un formulaire

■ La suite :

```
// Si la requête est en POST
if ($request->isMethod('POST')) {
    // On fait le lien Requête <-> Formulaire
    // À partir de maintenant, la variable $advert contient les valeurs entrées dans le
    formulaire par le visiteur
    $form->handleRequest($request);

    // On vérifie que les valeurs entrées sont correctes
    // (Nous verrons la validation des objets en détail dans le prochain chapitre)
    if ($form->isValid()) {
        // On enregistre notre objet $advert dans la base de données, par exemple
        $em = $this->getDoctrine()->getManager();
        $em->persist($advert);
        $em->flush();

        $request->getSession()->getFlashBag()->add('notice', 'Annonce bien enregistrée.');

        // On redirige vers la page de visualisation de l'annonce nouvellement créée
        return $this->redirectToRoute('oc_platform_view', array('id' => $advert->getId()));
    }
}

// cas d'une requete GET ou de champs non valides : on réaffiche le formulaire
return $this->render('OCPlatformBundle:Advert:add.html.twig', array(
    'form' => $form->createView(),
));
}
```

Options sur les formulaires

- On peut rendre des champs facultatifs, par exemple :

```
$formBuilder->add('published', CheckboxType::class,  
array('required' => false))
```

- On peut également afficher des valeurs par défaut

```
// On crée un objet Advert  
$advert = new Advert();  
  
// On place un texte par défaut  
$advert->setTitle("Nouvelle annonce");
```

Editer une annonce

- On peut aussi modifier une annonce déjà enregistrée en base de données (`editAction()`):

```
public function editAction($id, Request $request)
{
    $em = $this->getDoctrine()->getManager();

    // Récupération d'une annonce déjà existante, d'id $id.
    $advert = $this->getDoctrine()->getManager()-
    >getRepository('OCPlatformBundle:Advert')->find($id);

    // Et on construit le FormBuilder avec cette instance de l'annonce, comme
    précédemment
    $formBuilder = $this->get('form.factory')->createBuilder(FormType::class,
    $advert);

    // On ajoute les champs de l'entité que l'on veut à notre formulaire
    $formBuilder->add('date',      DateType::class);
    $formBuilder->add('title',     TextType::class);
    $formBuilder->add('content',   TextareaType::class);
    $formBuilder->add('author',    TextType::class);
    $formBuilder->add('published', CheckboxType::class);
    $formBuilder->add('save',      SubmitType::class);
    // À partir du FormBuilder, on génère le formulaire
    $form = $formBuilder->getForm();
```

Editer une annonce

- La suite ..

```
if ($request->isMethod('POST') && $form->handleRequest($request)->isValid())  
{  
    // Inutile de persister ici, Doctrine connaît déjà notre annonce  
    $em->flush();  
  
    $request->getSession()->getFlashBag()->add('notice', 'Annonce bien  
modifiée.');
```



```
    return $this->redirectToRoute('oc_platform_view', array('id' => $advert-  
>getId()));  
}
```



```
    return $this->render('OCPlatformBundle:Advert:edit.html.twig', array('form'  
=> $form->createView(), 'advert' => $advert));  
}
```

Aspect du formulaire

- Symfony contient un thème pour prendre en compte Bootstrap avec les formulaires.
- Pour l'utiliser, modifiez le fichier `app/config/config.yml`

```
# app/config/config.yml

twig:
  form_themes:
    - 'bootstrap_3_layout.html.twig'
```

Ajouter une annonce

Formulaire d'annonce

Date

Nov 14 2017

Title

Nouvelle annonce

Content

Author

☒ Published

Save

Attention : cette annonce sera ajoutée directement sur la page d'accueil après validation du formulaire.