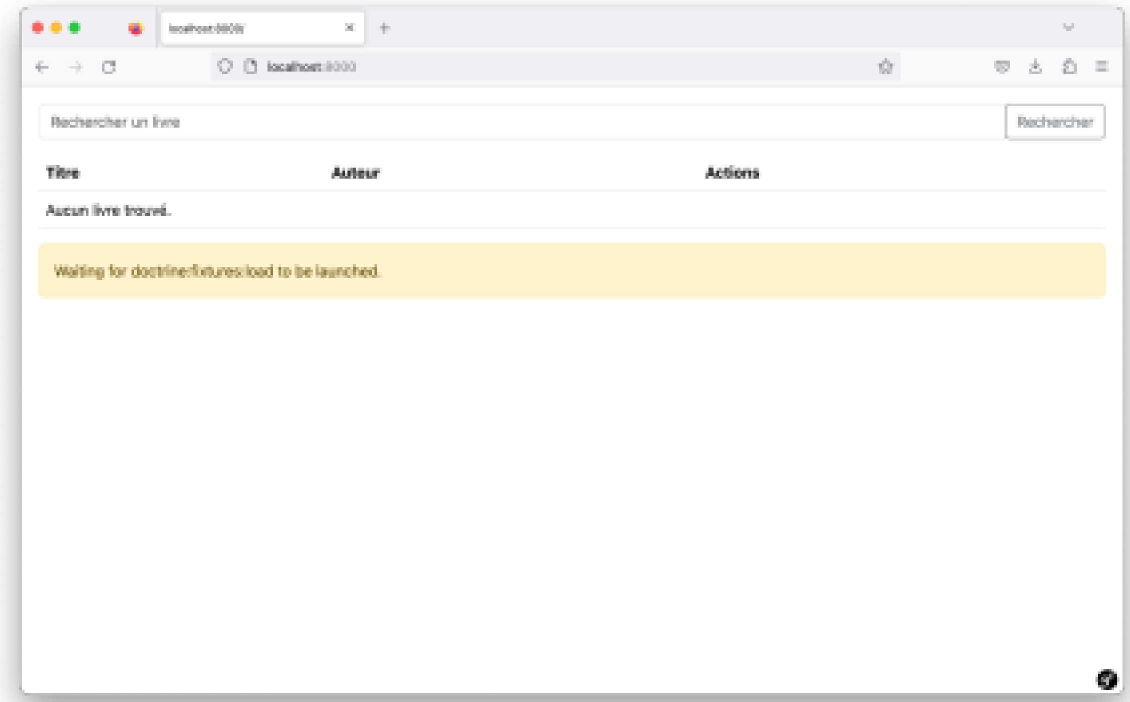
TP Symfony - Upload de fichiers & anonymisation

Préambule

Télécharger et tester sous Visual Studio Code le bon fonctionnement du projet de test en lançant le serveur Symfony local.



A ce stade, nous devons disposer d'une librairie vide de données.

1- Intégration des données de test

1.1 Lancer la commande de chargement des fixtures (données de test). Quelle commande avez-vous exécutée?

`bin/console doctrine:fixtures:load`

1.2 Relancer la commande et observer l'évolution des id de la table Book et Author. Que se passe-t-il?

Après avoir relancé la commande, la base de données est purgée (vidée) et les fixtures y sont chargées. Il n'y à pas d'évolution visuelle au niveau des id de la table Book et Author.

1.3 Dans la classe ***AppFixtures***, trouver et mettre en commentaire la ligne suivante:

src/DataFixtures/AppFixtures.php

```
$this->clearTablesAndAutoincrement($manager);
```

Relancer le chargement des fixtures. Que se passe-t-il?

Cette fois-ci, la commande supprime les données de la database et incrémente les données ajoutées par les fixtures.

1.4 Ajouter l'option "`--purge-with-truncate`" à la commande. Cette option purge les données en utilisant la méthode TRUNCATE, ce qui est plus rapide que la suppression standard et réinitialise également tous les auto-incréments.

Est-ce que cela fonctionne? Faire une recherche et expliquer pourquoi.

La nouvelle commande n'a pas fonctionné comme prévu.
Pour que la commande fonctionne correctement, il faut décommenter la ligne de code « `$this->clearTablesAndAutoincrement($manager);` ».

1.5 Sur la base du TP « ***Symfony - authentification et administration*** », créer un dashboard d'administration. Quelles commandes doit-on lancer?
Note: pour les besoins de simplification de ce TP, nous ne gérons pas les utilisateurs. L'accès à l'admin se fera sans identification.

1. `composer require admin`
2. `bin/console make:admin:dashboard`
    - `Which class name do you prefer for your Dashboard controller? :`
      `>` **`DashboardController`**
    - `In which directory of your project do you want to generate "DashboardController"? :`
      `>` **`src/Controller/Admin/`**

1.6 Sur la base du TP « ***Symfony - CRUD: Create, Read, Update, Delete*** » Créer les CRUD pour gérer la liste des livres et des auteurs. Vous ferez en sorte de gérer le lien livre/auteur sous forme d'une liste (**AssociationField**). Détailler les actions menées.

1.

`src/Controller/Admin/DashboardController.php`

```php
<?php

namespace App\Controller\Admin;

[...]
use App\Entity\Book;

class DashboardController extends AbstractDashboardController
{
    #[Route('/admin', name: 'admin')]
    public function index(): Response
    {
        return $this->render('admin/index.html.twig');
    }

    [...]

    public function configureMenuItems(): iterable
    {
        yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');

        yield MenuItem::linkToCrud('Books', 'fas fa-list', Book::class);
    }
}
```

2. **bin/console make:admin:crud**

- Which Doctrine entity are you going to manage with this CRUD controller? :
  [0] App\Entity\Author
  [1] App\Entity\Book
  > **1**
- Which directory do you want to generate the CRUD controller in? [src/Controller/Admin/] :
  > **src/Controller/Admin/**
- Namespace of the generated CRUD controller [App\Controller\Admin] :
  > **App\Controller\Admin**

3.

templates/admin/index.html.twig

```
{% extends '@EasyAdmin/page/content.html.twig' %}
```

4.

src/Controller/Admin/BookCrudController.php

```
<?php

namespace App\Controller\Admin;

use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractCrudController;
use EasyCorp\Bundle\EasyAdminBundle\Field\TextField;

// fichiers d’en-tête ajoutés
use App\Entity\Book;
use EasyCorp\Bundle\EasyAdminBundle\Field\AssociationField;

class BookCrudController extends AbstractCrudController
{
   public static function getEntityFqcn(): string
   {
       return Book::class;
   }

   public function configureFields(string $pageName): iterable
   {
       return [
           //IdField::new('id'),
           TextField::new('title'),
           TextField::new('category'),
           AssociationField::new('author'),
           TextField::new('description'),
       ];
   }
}
```

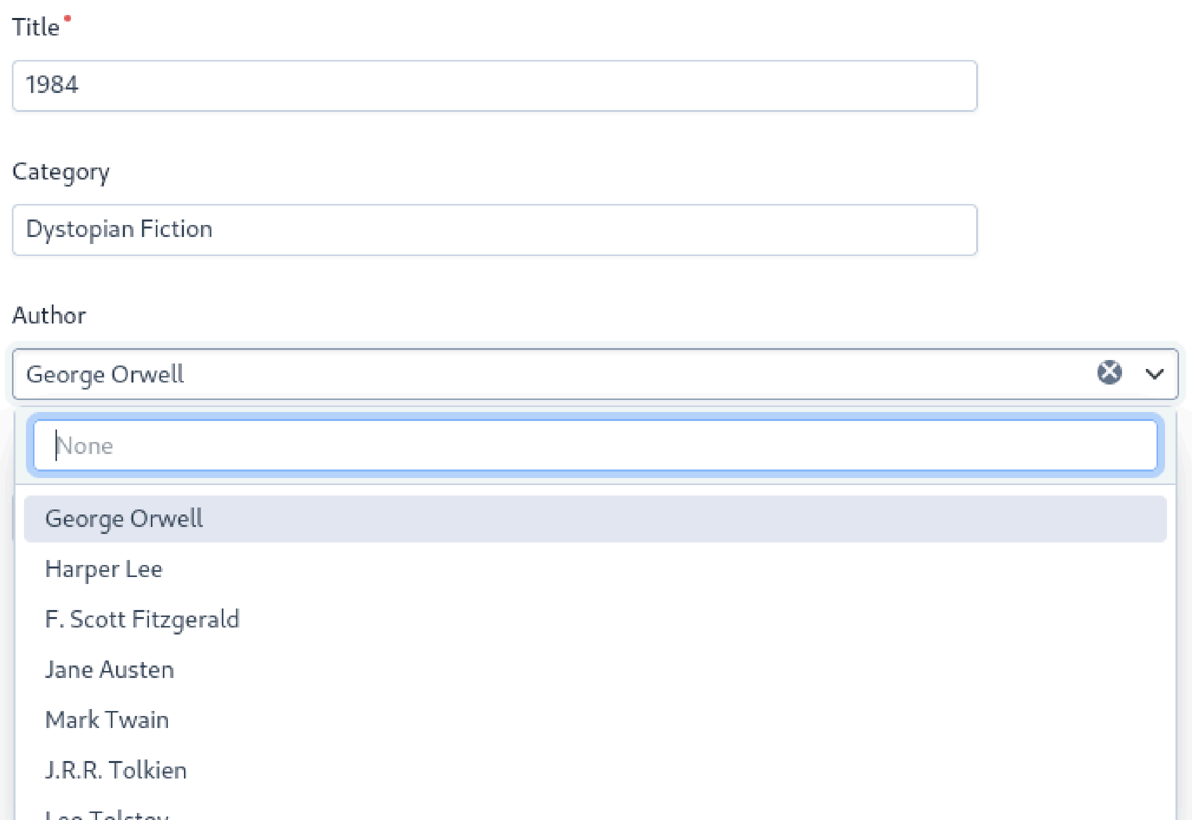
src/Entity/Author.php

```php
<?php

namespace App\Entity;

use App\Repository\AuthorRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: AuthorRepository::class)]
class Author
{
    [...]

    // fonction permettant l'utilisation de AssociationField
    public function __toString()
    {
        return $this->firstname . " " . $this->lastname;
    }
}
```

Book

Title	Category	Author
1984	Dystopian Fiction	George Orwell
To Kill a Mockingbird	Southern Gothic Fiction	Harper Lee
The Great Gatsby	Tragedy	F. Scott Fitzgerald
Pride and Prejudice	Classic Romance	Jane Austen
Adventures of Huckleberry Finn	Picaresque Novel	Mark Twain
The Hobbit	Fantasy	J.R.R. Tolkien
War and Peace	Historical Fiction	Leo Tolstoy
Great Expectations	Bildungsroman	Charles Dickens
Frankenstein	Gothic Novel	Mary Shelley
Moby Dick	Adventure Fiction	Herman Melville

Edit Book



2- Upload d’une page de couverture.

Maintenant que nous disposons d’un site avec un outil d’administration opérationnel, nous souhaitons enrichir notre entité Book avec une Cover (Couverture).

Pour nous faciliter la tâche nous allons exploiter le composant VichUploaderBundle…

```
composer require vich/uploader-bundle
```

Do you want to execute this recipe? **Y** (ou "**P**" pour "**permanant**")

Pour commencer, nous devons ensuite modifier le fichier de configuration du bundle

config/packages/vich_uploader.yaml

```yaml
vich_uploader:
    db_driver: orm # ou mongodb ou propel, etc.
    mappings:
        book_cover:
                uri_prefix: /uploads/book_covers
                upload_destination:
                '%kernel.project_dir%/public/uploads/book_covers'
                namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
                delete_on_update: true
                delete_on_remove: true
```

Note: Créer dans la foulée le répertoire **public/uploads/book_covers**. Puis, nous devons éditer **avec soin** notre entité Book. **Ne pas oublier la migration !**

Commencer par ajouter un attribut **updatedAt** (datetime) et **coverName** (string) via ***make:entity***. Le type File n’étant pas reconnu par Doctrine, nous devons ajouter l’attribut **coverFile** manuellement. Voici un exemple d’entité…

src/Entity/Book.php

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

use App\Repository\BookRepository;
use Doctrine\DBAL\Types\Types;
use Doctrine\ORM\Mapping as ORM;
use Vich\UploaderBundle\Mapping\Annotation as Vich;
use Symfony\Component\HttpFoundation\File\File;
use Symfony\Component\Validator\Constraints as Assert;

#[Vich\Uploadable]
#[ORM\Entity(repositoryClass: BookRepository::class)]
class Book
{
```

```php
    // ...

    #[ORM\Column(length: 255, nullable: true)]
    private ?string $coverName = null;

    #[Assert\File(mimeTypes: ["image/*"])]
    #[Vich\UploadableField(mapping: 'book_cover', fileNameProperty:
    'coverName')]
    private ?File $coverFile = null;

    // ...

    public function setCoverFile(?File $image = null): void
    {
        $this->coverFile = $image;

        if (null !== $image) {
            // It is required that at least one field changes for doctrine
            // otherwise the event listeners won't be called and the file is
            lost
            $this->updatedAt = new \DateTimeImmutable();
        }
    }

    public function getCoverFile(): ?File
    {
        return $this->coverFile;
    }

    public function setCoverName(?string $coverName): void
    {
        $this->coverName = $coverName;
    }

    public function getCoverName(): ?string
    {
        return $this->coverName;
    }

    // ...
}
```

Enfin, enrichir la méthode ***configureFields()*** de classe **BookCrudController** :

src/Controller/Admin/BookCrudController.php

```
public function configureFields(string $pageName): iterable
    {
        return [
            // Pour afficher l'image dans les actions d'index et de
            détail.
            ImageField::new('coverName')
            ->setBasePath('/uploads/book_covers')
            ->onlyOnIndex(),


            // … Les autres champs à afficher


           // Pour le formulaire de téléchargement dans les actions de
            // création et d'édition.
            TextField::new('coverFile')
                ->setFormType(VichImageType::class)
                ->setRequired(false)
                ->onlyOnForms(),
        ];
    }
```

2.1 Après avoir contrôlé le bon fonctionnement de l'ajout/suppression de cover, faire une recherche sur le namer: `Vich\UploaderBundle\Naming\UniqidNamer`
Expliquer son principe et détailler **les autres namers** disponibles.
Le bundle utilise des "namer" pour nommer les fichiers et les répertoires qu'il enregistre sur le système de fichiers.
Un namer met en œuvre l'interface **Vich\UploaderBundle\Naming\NamerInterface**. Il faut utiliser un des namers fournis ou en implémenter un personnalisé.

- **`UniqidNamer`** renomme les fichiers téléchargés en utilisant un identifiant unique pour le nom tout en conservant l'extension. En utilisant ce nom, "**foo.jpg**" sera téléchargé sous la forme "**0eb3db03971550eb3b0371.jpg**".

- **`OrignameNamer`** renomme les fichiers téléchargés en utilisant un identifiant unique comme préfixe du nom de fichier et en conservant le nom et l'extension d'origine. En utilisant ce nom, "**foo.jpg**" sera téléchargé sous la forme "**50eb3db039715_foo.jpg**".

- **`PropertyNamer`** utilisera une propriété ou une méthode pour nommer le fichier. Il faut spécifier quelle propriété sera utilisée :

```
vich_uploader:
   # ...
   mappings:
       products:
           upload_destination: products_fs
           namer:
               service: Vich\UploaderBundle\Naming\PropertyNamer
               options: { property: 'slug' } # supposing that the object contains a "slug" property or a "getSlug" method
```

- **`HashNamer`** utilise un hachage de chaîne aléatoire pour nommer le fichier. on peut également spécifier l'algorithme de hachage "`algorithm`" et la longueur "`length`" du résultat du fichier :

```
vich_uploader:
   # ...
   mappings:
       products:
           upload_destination: products_fs
           namer:
               service: Vich\UploaderBundle\Naming\HashNamer
               options: { algorithm: 'sha256', length: 50 }
```

- **`Base64Namer`** génère une chaîne aléatoire décodable en base64 pour nommer le fichier. Vous pouvez spécifier la longueur "`length`" de la chaîne aléatoire. En utilisant ce nom, "**foo.jpg**" sera téléchargé sous la forme "**6FMNgvkdUs.jpg**".

- **`SmartUniqueNamer`** renomme les fichiers téléchargés en ajoutant un identifiant unique fort au nom d'origine, tout en appliquant une translittération. En utilisant ce nommage, "**a Strange name.jpg**" sera téléchargé sous la forme "**a-strange-name-0eb3db03971550eb3b0371.jpg**".

- **`SlugNamer`** ne fait que translittérer le fichier téléchargé. Il cherche ensuite si ce nom existe déjà et, si c'est le cas, il ajoutera un numéro progressif (pour assurer l'unicité).
  Ce namer est utile lorsque l’on veut garder les noms aussi proches que possible des noms originaux, mais il est également limité à des situations simples (par exemple, lorsque vous utilisez une seule entité mappée).
  Pour l'utiliser, il suffit de spécifier le service pour l'option de configuration "`namer`" de votre mappage :

```
# config/services.yaml
services:
   Vich\UploaderBundle\Naming\SlugNamer:
       public: true
       arguments:
           $service: '@App\Repository\MyFileRepository'
           $method: findOneByPath
```

2.2 En quoi l’usage du renommage de fichier peut-être très utile pour l’anonymisation de document?

Le renommage de fichier peut-être très utile pour l’anonymisation de document, puisqu’il peut encoder le nom du fichier, ajouter ou remplacer des caractères, voire remplacer entièrement le nom du fichier.

2.3 Éditer les twig de visualisation de l’accueil et de la fiche détail pour afficher la Cover. Quelles sont les modifications effectuées?

templates/book/index.html.twig

```
[...]
   <div class="table-responsive">
       <table class="table">
           <thead>
               <tr>
                   <th>Première de couverture</th>
                   <th>Titre</th>
                   <th>Auteur</th>
                   <th>Actions</th>
               </tr>
           </thead>
           <tbody>
           {% for book in books %}
               <tr>
                   <td><img src="/uploads/book_covers/{{ book.coverName
}}" heigth=260 width=200/></td>
                   <td>{{ book.title }}</td>
                   <td>{{ book.author.firstname ~ ' ' ~
book.author.lastname }}</td>
                   <td>
                       <a href="{{ path('book_show', {'id': book.id})
}}" class="btn btn-primary">Plus d'infos</a>
                   </td>
               </tr>
           {% else %}
               <tr>
                   <td colspan="3">Aucun livre trouvé.</td>
               </tr>
           {% endfor %}
           </tbody>
       </table>
   </div>
[...]
```

templates/book/show.html.twig

```
{% extends 'base.html.twig' %}

{% block title %}Détails du livre{% endblock %}

{% block body %}
<div class="content-wrapper">
 <h1>{{ book.title }}</h1>
 <p><strong>Auteur :</strong> {{ book.author ? book.author.firstname ~ '
' ~ book.author.lastname : 'Non attribué' }}</p>
 <p><strong>Description :</strong> {{ book.description }}</p>
 <p><strong>Catégorie :</strong> {{ book.category }}</p>

 <p><strong>Première de couverture :</strong><br><img
src="/uploads/book_covers/{{ book.coverName }}" heigth=500
width=375/></p>

 <a href="{{ path('app_book') }}" class="btn btn-secondary">Retour à la
liste</a>
</div>
{% endblock %}
```