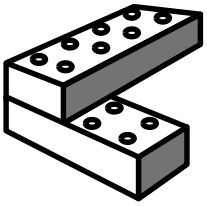

Travaux Pratiques : Enoncés
TP_ReviewPhoto_Partie_08

Objectif : **Consolidation sur le framework Symfony 6**
Upload des photos

Il nous faut faire en sorte que l'utilisateur identifié puisse uploader une photo. On pourrait faire tout le code uniquement avec le framework Symfony :

Voir doc : https://symfony.com/doc/current/controller/upload_file.html

Mais comme il est indiqué dans cette documentation, il est plus simple d'utiliser une bibliothèque externe « VichUploaderBundle ».

Suivre le lien à partir de la documentation ci-dessus.

Réalisations :

1 : Installation du bundle « VichUploaderBundle» :

- **Entrer la commande :**

  composer require vich/uploader-bundle

Répondez « yes » à la question « faites-vous confiance »

Depuis Symfony 4 vous n'avez rien d'autre à faire lors de l'installation. Cette commande à :

- **Ajouter la référence du bundle dans « config/bundles.php »**
- **Créer un fichier de configuration « config/packages/vich_upload.yaml » qui précise que le bundle utilisera un orm.**

Reste à paramétrer ce bundle pour notre utilisation.

2 : Paramétrage du « VichUploaderBundle »

Il faut modifier le fichier de configuration « vich_uploader.yaml »

- **Ajouter une clé « mappings » avec une sous clé donnant le nom de « mappage »**
  - **choisir de l'appeler « uploaded_photo »**
- **Sous cette clé, il faut 3 autres clés :**
  - **« uri_prefix » qui précise l'uri web des fichiers images**
    - **choisir « /uploaded/photos**
  - **« upload_destination » qui donne le chemin windows du dossier des photos**
    - **choisir « %kernel.project_dir%/public/uploaded/photos »**
  - **« namer » qui précise un « namer » permettant de renommer les fichiers uploadés (il est obsolète de laisser le nom réel de l'image uploadée)**

- **choix entre plusieurs namer (voir doc https://github.com/dustin10/VichUploaderBundle/blob/master/docs/namers.md)**
- **Choisir « Vich\UploaderBundle\Naming\SmartUniqidNamer »**

Le fichier de configuration sera donc : (respecter bien les 4 espaces devant chaque sous clé)

```
vich_uploader:
    db_driver: orm

    mappings:
        uploaded_photo:
            uri_prefix: /assets/photos
            upload_destination: '%kernel.project_dir%/public/assets/photos'
            namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
```

Il faut relier notre « mappage » à notre entité « Photo »

3 : Modification de l'entité « Photo »

- **Il faut ajouter une annotation à notre classe :**

```
use Symfony\Component\HttpFoundation\File\File;
use Vich\UploaderBundle\Mapping\Annotation as Vich;

#[ORM\Entity]
#[Vich\Uploadable]
class Photo
{…
}
```

- **Ajouter trois propriétés après le « title »:**
  - **« imageName » qui sera persistée en base dans un champ de type « string » annotée :**

```
#[ORM\Column(length: 255)]
private ?string $imageName = null;
```

  - **« imageFile » qui ne sera de type « File » (voir use) non persistée mais annotée comme suit :**

```
#[Vich\UploadableField(mapping: 'uploaded_photos',
fileNameProperty: 'imageName', size: 'imageSize')]
private ?File $imageFile = null;
```

  - **« imageSize » qui sera persistée en base dans un champ de type « integer » annotée :**

```
#[ORM\Column(type: 'integer')]
private ?int $imageSize = null;
```

Le code de ces 3 propriétés est donc le suivant :

```
#[Vich\UploadableField(mapping: 'uploaded_photo',  fileNameProperty: 'imageName', size: 'imageSize')]
private ?File $imageFile = null;

#[ORM\Column(length: 255)]
private ?string $imageName = null;

#[ORM\Column(type: 'integer')]
private ?int $imageSize = null;
```

- **Créer les getteurs et setteurs pour ces propriétés :**

- **Modifier la méthode setImageFile() qui valorisera la propriété « post_at ». ceci permettra à Symfony de voir qu'une propriété persistée a changé et qu'il doit updater la photo en base.**

```
public function setImageFile(?File $imageFile = null): void {
    $this->imageFile = $imageFile;

    if (null !== $imageFile ){
        $this->post_at = new \DateTimeImmutable ();
    }
}
```

Attention : vérifier que le retour de la méthode getTitle() peut être null : Ajouter un « ? » devant le type du retour

```
public function getTitle(): ?string
 { … }
```

Une fois cette configuration terminée :

- **Vider le cache de Symfony par la commande**
  php bin/console cache:clear
- **Faire le fichier de migration**
  php bin/console make:migration
- **Valider cette migration**
  php bin/console doctrine:migrations:migrate

2 : Création du formulaire de saisie d'une photo :

- **Entre la commande :** *symfony console make:form*
  - **Nom : NewPhotoType**
  - **Classe : Photo**

3 : Modification de la méthode « buildForm » de la classe « NewPhotoType »

- **Supprimer les champs**

- **« post_at » qui est valorisé dans l'entité » « Photo »**
- **« user » qui sera valoriser dans le contrôleur**
- **« imageSize »**

- **Modification des champs :**

->add('title') devient
->add('title', TextType::class, ['label' => 'Nom de la photo'])
->add('imageName') devient
->add('imageFile', VichImageType::class, ['label' => 'Votre photo'])

Attention : c'est bien le champ « imageFile » et non « imageName »

Dans le dossier « Form » créé un sous-dossier « Photo » et placer y le fichier « NewPhotoType.php », modifier le namespace « App\Form\Photo » de cette classe.

4 : Création de la méthode « new() » dans le « PhotoController » :

Modifier tout d'abord la route de la méthode « show() » comme suit sinon la route pour la nouvelle photo sera aussi matchée par celle de « show() »

```
@Route("/photo/show/{id}", name="photo.show")
```

Cette méthode ressemble à la méthode « new() » de la classe « UserController »

L'annotation pour la route sera :

```
@Route("/photo/new", name="photo.new")
```

Dans cette méthode nous allons suivre la même logique :

Création d'un nouvel objet Photo :
$photo = new Photo() ;
Valorisation de la propriété « user » par l'utilisateur identifié :
$photo->setUser($this->getUser());
Création d'un objet formulaire de classe « UserType » :
$form = $this->createForm(PhotoType::class, $photo);
Valorisation de ses champs par les valeurs de la requête http :
$form->handleRequest($request);
Si (formulaire soumis et valide) Alors

On initialise la propriété « user » par l'utilisateur connecté :

```
$photo->setUser($this->getUser());
```

On initialise la date du post :

```
$photo->setPostAt(new \DateTimeImmutable());
```

Grace à l'entityManager récupéré par l'injection de dépendance :

On persiste :

```
$entityManager m->persist($photo);
```

On flush

```
$entityManager ->flush();
```

On retourne à la vue « home » :

```
return $this->redirectToRoute('photo.list');
```

Sinon

Affichage du formulaire :

```
return $this->render('photo/new.html.twig', [
        'form' => $form->createView()
        ]);
```

Finsi

5 : Création de la vue « new.html.wtig » dans le dossier « templates/photo »

Le code peut être :

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

{% block title 'Uploader une photo' %}

  {% block body %}
    <div class="container mt-4">

    {{ form_start(form) }}
      <div class="row">
        <div class="col-md-4">{{ form_row(form.title) }}</div>
       </div>
      <div class="row">
        <div class="col-md-2">{{ form_row(form.imageFile) }}</div>
      </div>
      <button class="btn btn-primary">Poster</button>
    {{ form_end(form) }}

    </div>
  {% endblock %}
```

6 : Affichage des photos dans les vues « list.html.twig » et « show.html.twig »

- **Dans ces deux vues, modifier la propriété « src » de la balise « img » comme suit :**

```
<img class="card-img-top" src="{{ vich_uploader_asset(photo, 'imageFile') }}"
                          alt="photo de {{ photo.user.pseudo }}">
```

7 : tester