Chapitre 9 – L’upload de fichiers

Introduction

Dans le chapitre précédent, nous avons donné des droits à des types d’utilisateurs. De plus, chaque utilisateur peut maintenant consulter son profil.
Dans ce chapitre, les utilisateurs vont pouvoir envoyer des fichiers sur le site.

Création du contrôleur

Nous allons créer une nouvelle page qui sera accessible avec l’url suivante :

```
/profil-ajout-fichier
```

Pour cela, nous allons créer le contrôleur « Fichier ».

```
php bin/console make:controller
```

src/Controller/FichierController.php

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class FichierController extends AbstractController
{
    #[Route('/profil-ajout-fichier', name: 'ajout-fichier')]
    public function index(): Response
    {
        return $this->render('fichier/ajout-fichier.html.twig', [

        ]);
```

```
    }
}
```

Remarque : l'url commence par « profil », ce qui signifie que seules les personnes connectées pourront se rendre sur cette page.

Création de la vue

ajout-fichier.html.twig

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

{% block title %}
        {{parent()}}Ajouter un fichier
{% endblock %}

{% block body %}
        <div class="container-fluid">
                <h1 class="text-center text-primary mt-4 pt-4 display-1 fw-bold">
                        Ajouter un fichier</h1>
    </div>
{% endblock %}
```

Création du formulaire

Nous allons créer le formulaire « AjoutFichierType ».

```
php bin/console make:form
```

src/Form/AjoutFichierType.php

```
<?php

namespace App\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\FileType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class AjoutFichierType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('fichier', FileType::class, array('label' => 'Fichier à télécharger'))
            ->add('envoyer', SubmitType::class, ['attr' => ['class'=> 'btn bg-primary text-white m-4' ], 'row_attr' => ['class' => 'text-center'],])
```

```
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            // Configure your form options here
        ]);
    }
}
```

Remarque : nous utilisons pour la première fois le composant « FileType » permettant de sélectionner un fichier sur votre système afin de l'envoyer. N'oubliez pas d'ajouter le « use » correspondant en haut du fichier.

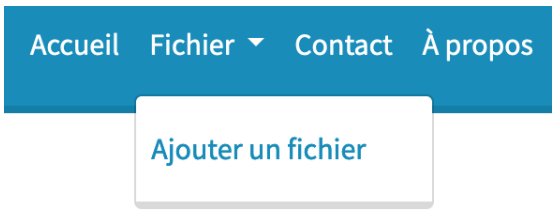
Modification du menu

Nous allons modifier le menu afin qu'un utilisateur connecté (peu importe son rôle), puisse se rendre sur notre nouvelle page.

template/base.html.twig

```
{% if is_granted('IS_AUTHENTICATED_FULLY') %}
    <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle text-white" href="#" id="navbarScrollingDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
                                                    Fichier
          </a>
          <ul class="dropdown-menu" aria-labelledby="navbarScrollingDropdown">
            <li class="nav-item">
              <a class="nav-link text-primary" href="{{path('ajout-fichier')}}">Ajouter un fichier
              </a>
            </li>
          </ul>
    </li>
{% endif %}
```

Commentaire :
IS_AUTHENTICATED_FULLY permet de savoir si vous êtes connecté ou pas.



Ajout du formulaire dans le contrôleur

src/Controller/FichierController.php

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use App\Form\AjoutFichierType;

class FichierController extends AbstractController
{
    #[Route('/profil-ajout-fichier', name: 'ajout-fichier')]
    public function index(): Response
    {
        $form = $this->createForm(AjoutFichierType::class);
        return $this->render('fichier/ajout-fichier.html.twig', [
            'form'=> $form->createView()
        ]);
    }
}
```

Ajout du formulaire dans la vue

templates/fichier/ajout-fichier.html.twig

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

{% block title %}
        {{parent()}}Ajouter un fichier
{% endblock %}

{% block body %}
        <div class="container-fluid">
                <h1 class="text-center text-primary mt-4 pt-4 display-1 fw-bold">
                        Ajouter un fichier</h1>
        <div class="row justify-content-center">
                        <div class="col-12 col-md-8 bg-white p-4 m-0 text-primary">
        {{form(form)}}
        </div>
```

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

Voici le rendu de l'interface :

Création du répertoire qui accueillera les fichiers

À la racine du projet, créez un répertoire « uploads ». À l'intérieur de celui-ci, créez un répertoire « fichiers ». Enfin, donnez les droits suivants au répertoire « uploads » :

```
chmod -R 777 uploads
```

Le répertoire « upload » nous permettra de stocker des fichiers envoyés par l'utilisateur. Le répertoire « fichiers » contiendra les fichiers qui seront partagés entre utilisateurs.

Dans le fichier « services.yaml », créez une clé permettant de configurer le chemin vers votre répertoire « fichiers ».

config/services.yaml

```
parameters:
    file_directory: '%kernel.project_dir%/uploads/fichiers'
```

Commentaire :
La clé est file_directory. Nous lui mettons le chemin où se situe l'application à l'aide de « %kernel.project_dir% » et nous mettons à la suite les répertoires jusqu'à « fichiers ».

src/Controller/FichierController.php

```
<?php
```

```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\String\Slugger\SluggerInterface;
use App\Form\AjoutFichierType;

class FichierController extends AbstractController
{
    #[Route('/profil-ajout-fichier', name: 'ajout-fichier')]
    public function index(Request $request, SluggerInterface $slugger): Response
    {
        $form = $this->createForm(AjoutFichierType::class);
        if($request->isMethod('POST')){
            $form->handleRequest($request);
            if ($form->isSubmitted() && $form->isValid()){
                $fichier = $form->get('fichier')->getData();

                if($fichier){
                 $nomFichier = pathinfo($fichier->getClientOriginalName(), PATHINFO_FILENAME);
                 $nomFichier = $slugger->slug($nomFichier);
                 $nomFichier = $nomFichier.'-'.uniqid().'.'.$fichier->guessExtension();
                    try{
                        $fichier->move($this->getParameter('file_directory'), $nomFichier);
                        $this->addFlash('notice', 'Fichier envoyé');
                    }
                    catch(FileException $e){
                        $this->addFlash('notice', 'Erreur d\'envoi');
                    }
                }
                return $this->redirectToRoute('ajout-fichier');
            }
        }
        return $this->render('fichier/ajout-fichier.html.twig', [
            'form'=> $form->createView()
        ]);
    }
}
```

Commentaires :
SluggerInterface $slugger // nous récupérons une variable de type « SluggerInterface » qui nous permettra de formater une chaine de caractères en y retirant les espaces et les caractères spéciaux.
$nomFichier = pathinfo($fichier->getClientOriginalName(), PATHINFO_FILENAME); // nous récupérons le nom du fichier sans son extension. Le nom original du fichier se trouve dans getClientOriginalName.
pathinfo est une fonction qui permet de récupérer l'élément que nous souhaitons dans
$nomFichier = $slugger->slug($nomFichier); // Nous appliquons le slug afin de formater le nom du fichier.

$nomFichier = $nomFichier.'-'.uniqid().'.'.$fichier->guessExtension(); // Nous prenons le nom du fichier formaté auquel nous ajoutons un numéro unique et son extension. Son extension est récupérée à l'aide du MIME du fichier et non pas en lisant celle du fichier qui peut être modifiée facilement.
$fichier->move($this->getParameter('file_directory'), $nomFichier);
Nous déplaçons le fichier de la zone temporaire du serveur au répertoire « fichiers » que nous avons défini dans le fichier « services.yaml ». C'est la méthode « getParameter » qui permet de retrouver la clé « file_directory ».

Méthode	Description
getSize()	Récupère la taille du fichier en octets
guessExtension()	Récupère le type du fichier en lisant son MIME
getClientOriginalName()	Récupère le nom du fichier

Vous pouvez consulter la liste des différents Mimes : Mimes

```
<?php

namespace App\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\FileType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Validator\Constraints\File;

class AjoutFichierType extends AbstractType
{
   public function buildForm(FormBuilderInterface $builder, array $options): void
   {
      $builder
         ->add('fichier', FileType::class, array('label' => 'Fichier à télécharger',
         'constraints' => [
            new File([
               'maxSize' => '200k',
               'mimeTypes' => [
                  'application/pdf',
                  'application/x-pdf',
                  'image/jpeg',
                  'image/png',
               ],
               'mimeTypesMessage' => 'Le site accepte uniquement les fichiers PDF, PNG et JPG',
            ])
         ],))
           ->add('envoyer', SubmitType::class, ['attr' => ['class'=> 'btn bg-primary text-white m-4' ],
'row_attr' => ['class' => 'text-center'],])
      ;
```

```
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            // Configure your form options here
        ]);
    }
}
```

Commentaires :

```
'constraints' => [ // Nous ajoutons une contrainte sur le composant « FileType ».
            new File([ // La contrainte est de type File
                'maxSize' => '200k', // Nous limitons à 200ko la taille du fichier
                'mimeTypes' => [ // Nous mettons la liste des MIMES que nous acceptons
                    'application/pdf',
                    'application/x-pdf',
                    'image/jpeg',
                    'image/png',
                ],
```

Il est maintenant nécessaire de relier le fichier à un utilisateur, car pour l'instant, nous n'avons aucun moyen de savoir à qui appartient tel ou tel fichier. Pour cela, nous allons créer une nouvelle entité dont voici la structure :

Entité Fichier

Nom de l'attribut	Type	Signification
nomOriginal	string(taille de 255 caractères)	Nom du fichier
nomServeur	string(taille de 255 caractères)	Nom du fichier lorsqu'il est stocké sur le serveur
dateEnvoi	datetime	Date d'envoi sur le serveur
extension	string(taille de 4 caractères)	Extension du fichier
taille	float	Taille du fichier en octets
proprietaire	User	Propriétaire du fichier

```
php bin/console make:entity Fichier
```

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

 Field type (enter ? to see all types) [string]:
 > string
```

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

Add another property? Enter the property name (or press <return> to stop adding fields):
> nomServeur

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

Add another property? Enter the property name (or press <return> to stop adding fields):
> dateEnvoi

Field type (enter ? to see all types) [string]:
> datetime

Can this field be null in the database (nullable) (yes/no) [no]:
> no

Add another property? Enter the property name (or press <return> to stop adding fields):
> extension

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 3

Can this field be null in the database (nullable) (yes/no) [no]:
> no

Add another property? Enter the property name (or press <return> to stop adding fields):
> taille

Field type (enter ? to see all types) [string]:
> float

Can this field be null in the database (nullable) (yes/no) [no]:
> no

Add another property? Enter the property name (or press <return> to stop adding fields):
> proprietaire

```
 Field type (enter ? to see all types) [string]:
 > ManyToOne

 What class should this entity be related to?:
 > User

 Is the Fichier.proprietaire property allowed to be null (nullable)? (yes/no) [yes]:
 > no

 Do you want to add a new property to User so that you can access/update Fichier objects from it - e.g. $user->getFichiers()? (yes/no) [yes]:
 > yes

 A new property will also be added to the User class so that you can access the related Fichier objects from it.

 New field name inside User [fichiers]:
 > fichiers
```

Commentaires :

Add another property? Enter the property name (or press <return> to stop adding fields):
> proprietaire

Le propriétaire du fichier est une propriété à ajouter dans l'entité « Fichier ». Il s'agit d'un « User », il faut donc le relier à cette entité.

Un fichier n'a qu'un seul propriétaire, mais un propriétaire peut avoir plusieurs fichiers. Ainsi, de l'entité « Fichier » à « User », nous aurons une relation « ManyToOne ». Cela correspond à la cardinalité 1,1 que vous trouvez dans un MCD.

```
 Is the Fichier.proprietaire property allowed to be null (nullable)? (yes/no) [yes]:
 > no
```

Il sera obligatoire de relier un fichier à un « User », car il n'est pas normal d'avoir un fichier sans propriétaire.

```
Do you want to add a new property to User so that you can access/update Fichier objects from it - e.g. $user->getFichiers()? (yes/no) [yes]:
 > yes
```

Il sera possible de récupérer très facilement les fichiers d'un utilisateur, car nous créons une méthode « getFichiers() » dans l'entité « User ».

src/Entity/Fichier.php

```
<?php

namespace App\Entity;

use App\Repository\FichierRepository;
use Doctrine\ORM\Mapping as ORM;
```

```
/**
 * @ORM\Entity(repositoryClass=FichierRepository::class)
 */
class Fichier
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=50)
     */
    private $nomOriginal;

    /**
     * @ORM\Column(type="string", length=50)
     */
    private $nomServeur;

    /**
     * @ORM\Column(type="datetime")
     */
    private $dateEnvoi;

    /**
     * @ORM\Column(type="string", length=3)
     */
    private $extension;

    /**
     * @ORM\Column(type="float")
     */
    private $taille;

    /**
     * @ORM\ManyToOne(targetEntity=User::class, inversedBy="fichiers")
     * @ORM\JoinColumn(nullable=false)
     */
    private $proprietaire;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getNomOriginal(): ?string
    {
```

```
    return $this->nomOriginal;
}

public function setNomOriginal(string $nomOriginal): self
{
    $this->nomOriginal = $nomOriginal;

    return $this;
}

public function getNomServeur(): ?string
{
    return $this->nomServeur;
}

public function setNomServeur(string $nomServeur): self
{
    $this->nomServeur = $nomServeur;

    return $this;
}

public function getDateEnvoi(): ?\DateTimeInterface
{
    return $this->dateEnvoi;
}

public function setDateEnvoi(\DateTimeInterface $dateEnvoi): self
{
    $this->dateEnvoi = $dateEnvoi;

    return $this;
}

public function getExtension(): ?string
{
    return $this->extension;
}

public function setExtension(string $extension): self
{
    $this->extension = $extension;

    return $this;
}

public function getTaille(): ?float
{
    return $this->taille;
}
```

```
    public function setTaille(float $taille): self
    {
        $this->taille = $taille;

        return $this;
    }

    public function getProprietaire(): ?User
    {
        return $this->proprietaire;
    }

    public function setProprietaire(?User $proprietaire): self
    {
        $this->proprietaire = $proprietaire;

        return $this;
    }
}
```

Commentaires :
Nous retrouvons la propriété « proprietaire » avec ses accesseurs. Nous remarquons que la relation ManyToOne est indiquée au-dessus de la déclaration de l'attribut.

```
  * @ORM\ManyToOne(targetEntity=User::class, inversedBy="fichiers")
  * @ORM\JoinColumn(nullable=false)
private $proprietaire;
```

targetEntity fait référence à l'entité que nous souhaitons relier
inversedBy est la propriété dans l'entité «User» qui fait référence à l'entité Fichier.
nullable=false indique que le propriétaire est obligatoire pour ajouter un fichier.

src/Entity/User.php

```
<?php

namespace App\Entity;

use App\Repository\UserRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Security\Core\User\UserInterface;

/**
 * @ORM\Entity(repositoryClass=UserRepository::class)
 * @UniqueEntity(fields={"email"}, message="There is already an account with this email")
 */
class User implements UserInterface, PasswordAuthenticatedUserInterface
```

```
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     */
    private $email;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

    /**
     * @var string The hashed password
     * @ORM\Column(type="string")
     */
    private $password;

    /**
     * @ORM\Column(type="boolean")
     */
    private $isVerified = false;

    /**
     * @ORM\Column(type="datetime", nullable=true)
     */
    private $dateInscription;

    /**
     * @ORM\Column(type="string", length=30)
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=30)
     */
    private $prenom;

    /**
                 *     @ORM\OneToMany(targetEntity=Fichier::class,     mappedBy="proprietaire",
orphanRemoval=true)
     */
    private $fichiers;
```

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

public function getId(): ?int
{
    return $this->id;
}

public function getEmail(): ?string
{
    return $this->email;
}

public function setEmail(string $email): self
{
    $this->email = $email;

    return $this;
}

/**
 * A visual identifier that represents this user.
 *
 * @see UserInterface
 */
public function getUserIdentifier(): string
{
    return (string) $this->email;
}

/**
 * @deprecated since Symfony 5.3, use getUserIdentifier instead
 */
public function getUsername(): string
{
    return (string) $this->email;
}

/**
 * @see UserInterface
 */
public function getRoles(): array
{
    $roles = $this->roles;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'ROLE_USER';
```

```
    return array_unique($roles);
}

public function setRoles(array $roles): self
{
    $this->roles = $roles;

    return $this;
}

/**
 * @see PasswordAuthenticatedUserInterface
 */
public function getPassword(): string
{
    return $this->password;
}

public function setPassword(string $password): self
{
    $this->password = $password;

    return $this;
}

/**
 * Returning a salt is only needed, if you are not using a modern
 * hashing algorithm (e.g. bcrypt or sodium) in your security.yaml.
 *
 * @see UserInterface
 */
public function getSalt(): ?string
{
    return null;
}

/**
 * @see UserInterface
 */
public function eraseCredentials()
{
    // If you store any temporary, sensitive data on the user, clear it here
    // $this->plainPassword = null;
}

public function isVerified(): bool
{
    return $this->isVerified;
}
```

```
    public function setIsVerified(bool $isVerified): self
    {
        $this->isVerified = $isVerified;

        return $this;
    }

    public function getDateInscription(): ?\DateTimeInterface
    {
        return $this->dateInscription;
    }

    public function setDateInscription(?\DateTimeInterface $dateInscription): self
    {
        $this->dateInscription = $dateInscription;

        return $this;
    }

    public function getNom(): ?string
    {
        return $this->nom;
    }

    public function setNom(string $nom): self
    {
        $this->nom = $nom;

        return $this;
    }

    public function getPrenom(): ?string
    {
        return $this->prenom;
    }

    public function setPrenom(string $prenom): self
    {
        $this->prenom = $prenom;

        return $this;
    }

    /**
     * @return Collection|Fichier[]
     */
    public function getFichiers(): Collection
    {
        return $this->fichiers;
    }
```

```
    public function addFichier(Fichier $fichier): self
    {
        if (!$this->fichiers->contains($fichier)) {
            $this->fichiers[] = $fichier;
            $fichier->setProprietaire($this);
        }

        return $this;
    }

    public function removeFichier(Fichier $fichier): self
    {
        if ($this->fichiers->removeElement($fichier)) {
            // set the owning side to null (unless already changed)
            if ($fichier->getProprietaire() === $this) {
                $fichier->setProprietaire(null);
            }
        }

        return $this;
    }
}
```

Commentaires :

Nous remarquons que l'attribut « fichiers » a été créé automatiquement dans l'entité « Fichier ».

```
/**
* @ORM\OneToMany(targetEntity=Fichier::class, mappedBy="proprietaire",
orphanRemoval=true)
*/
private $fichiers;
```

Il a par conséquent créé l'association inverse « OneToMany ». La propriété « propriétaire » est celle créée dans l'entité « User ».

ornphanRemoval=true veut dire que si nous supprimons un utilisateur, nous supprimons tous les fichiers qu'il possède dans la table « Fichier ». Attention, cela ne veut pas dire que les fichiers physiques seront supprimés par la même occasion. Nous devrons réaliser un traitement supplémentaire.

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

Dans le constructeur, nous instancions une liste de fichiers vide. En programmation-objet, nous appelons cela une collection. Elle possède des méthodes permettant par exemple d'ajouter ou de supprimer une valeur. Chaque valeur de cette collection sera un fichier.

```
 /**
     * @return Collection|Fichier[]
     */
    public function getFichiers(): Collection
    {
        return $this->fichiers;
```

```
    }
```

La méthode « getFichiers » permet de récupérer la collection contenant tous les fichiers de l'utilisateur.

```
public function addFichier(Fichier $fichier): self
    {
        if (!$this->fichiers->contains($fichier)) {
            $this->fichiers[] = $fichier;
            $fichier->setProprietaire($this);
        }

        return $this;
    }
```

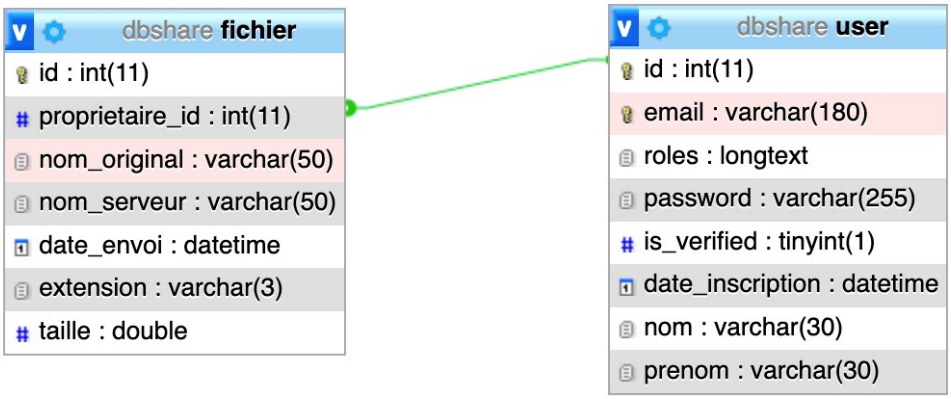
La méthode « addFichier » reçoit en paramètre un fichier et si la collection ne le possède pas déjà, elle l'ajoute. De plus, elle renseigne le propriétaire dans le fichier.

```
    public function removeFichier(Fichier $fichier): self
    {
        if ($this->fichiers->removeElement($fichier)) {
            // set the owning side to null (unless already changed)
            if ($fichier->getProprietaire() === $this) {
                $fichier->setProprietaire(null);
            }
        }

        return $this;
    }
```

La méthode « removeFichier » reçoit en paramètre le fichier à supprimer dans la collection. Elle indique également au fichier que le propriétaire n'est plus cet utilisateur en y mettant la valeur « null ».

```
php bin/console make:migration
```

```
php bin/console doctrine:migrations:migrate
```



Remarque :
Nous constatons que l'association apparait dans le concepteur de PhpMyAdmin, car les deux entités sont reliées.

src/Controller/FichierController.php

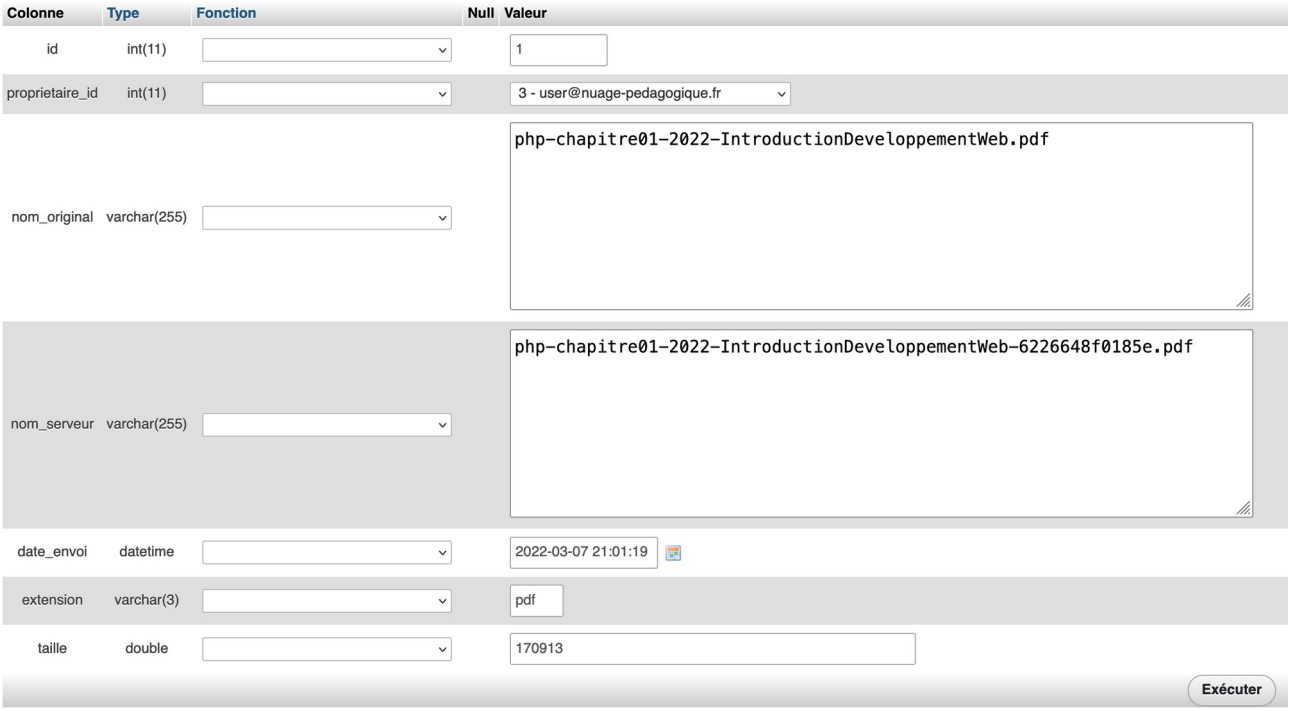
Mettre en haut du fichier le « use » permettant d'importer notre nouvelle entité :

```
use App\Entity\Fichier
```

```
#[Route('/profil-ajout-fichier', name: 'ajout-fichier')]
    public function index(Request $request, SluggerInterface $slugger): Response
    {
        $form = $this->createForm(AjoutFichierType::class);
        if($request->isMethod('POST')){
            $form->handleRequest($request);
            if ($form->isSubmitted() && $form->isValid()){

                $fichier = $form->get('fichier')->getData();

                if($fichier){
                 $nomFichier = pathinfo($fichier->getClientOriginalName(), PATHINFO_FILENAME);
                    $nomFichier = $slugger->slug($nomFichier);
                    $nomFichier = $nomFichier.'-'.uniqid().'.'.$fichier->guessExtension();
                    try{
                        $f = new Fichier();
                        $f->setNomServeur($nomFichier);
                        $f->setNomOriginal($fichier->getClientOriginalName());
                        $f->setDateEnvoi(new \Datetime());
                        $f->setExtension($fichier->guessExtension());
                        $f->setTaille($fichier->getSize());
                        $f->setProprietaire($this->getUser());

                        $fichier->move($this->getParameter('file_directory'), $nomFichier);
                        $this->addFlash('notice', 'Fichier envoyé');
                        $em = $this->getDoctrine()->getManager();
                        $em->persist($f);
                        $em->flush();

                    }
                    catch(FileException $e){
                        $this->addFlash('notice', 'Erreur d\'envoi');
                    }

                }
                return $this->redirectToRoute('ajout-fichier');
            }
        }
```

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

Remarques :



Nous mettons en mémoire une instance de la classe « Fichier » puis nous y mettons les données concernant le fichier physique : sa taille, son nom, etc.

src/Controller/FichierController.php

```
#[Route('/profil-ajout-fichier', name: 'ajout-fichier')]
    public function index(Request $request, SluggerInterface $slugger): Response
    {
        $form = $this->createForm(AjoutFichierType::class);
        if($request->isMethod('POST')){
            $form->handleRequest($request);
            if ($form->isSubmitted() && $form->isValid()){

                $fichier = $form->get('fichier')->getData();

                if($fichier){
                                        $nomFichier = pathinfo($fichier->getClientOriginalName(),
PATHINFO_FILENAME);
                    $nomFichier = $slugger->slug($nomFichier);
                    $nomFichier = $nomFichier.'-'.uniqid().'.'.$fichier->guessExtension();
                    try{
                        $f = new Fichier();
```

```
                $f->setNomServeur($nomFichier);
                $f->setNomOriginal($fichier->getClientOriginalName());
                $f->setDateEnvoi(new \Datetime());
                $f->setExtension($fichier->guessExtension());
                $f->setTaille($fichier->getSize());
                $f->setProprietaire($this->getUser());

                $fichier->move($this->getParameter('file_directory'), $nomFichier);
                $this->addFlash('notice', 'Fichier envoyé');

            }
            catch(FileException $e){
                $this->addFlash('notice', 'Erreur d\'envoi');
            }
            $em = $this->getDoctrine()->getManager();
            $em->persist($f);
            $em->flush();
        }
        return $this->redirectToRoute('ajout-fichier');
    }
}

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

templates/fichier/ajout-fichier.html.twig

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

{% block title %}
    {{parent()}}Ajouter un fichier
{% endblock %}

{% block body %}
    <div class="container-fluid">
        <h1 class="text-center text-primary mt-4 pt-4 display-1 fw-bold">
            Ajouter un fichier</h1>
        <div class="row justify-content-center">
            <div class="col-12 bg-white p-4 m-0 text-primary">
                {{form(form)}}
                </div>
            </div>
            <div class="row justify-content-center">
                <div class="col-12 bg-white p-4 m-0 text-primary">
                    <div class="table-responsive">
                        <table class="table caption-top">
                            <caption>Liste des fichiers</caption>
                            <thead>
```

```
                                                    <tr>
                                                        <th scope="col">Nom</th>

                                                        <th scope="col">Date</th>
                                                        <th
scope="col">Extension</th>
                                                        <th scope="col">Taille</th>
                                                    </tr>
                                                </thead>
                                                <tbody>
                                                    {% for f in app.user.fichiers | sort((a, b)
=> a.nomOriginal <=> b.nomOriginal) %}
                                                        <tr     class="{{     cycle(['table-
primary', 'table-secondary'], loop.index0) }}">

        <td>{{f.nomOriginal}}</td>

                                                            <td>{{f.dateEnvoi        |
date("d-m-Y à H:i:s") }}</td>
                                                            <td>
                                                                {% if f.extension
== 'pdf'%}
                                                                    <i
class="bi bi-file-earmark-pdf"></i>
                                                                {% else %}
                                                                    <i
class="bi bi-file-earmark-text"></i>
                                                                {% endif %}
                                                            </td>
                                                            {% set ko = f.taille /
1000 %}
                                                            <td>{{ko}}
                                                                ko</td>
                                                        </tr>
                                                    {% endfor %}
                                                </tbody>
                                            </table>
                                        </div>
                                    </div>
                                </div>
                            </div>
        {% endblock %}
```

Remarques :
{% for f in app.user.fichiers | sort((a, b) => a.nomOriginal <=> b.nomOriginal) %}
Nous parcourons l'ensemble des fichiers de l'utilisateur connecté. Nous utilisons le filtre « sort » afin de trier le résultat par ordre alphabétique sur le nom original du fichier.
{% set ko = f.taille / 1000 %} // Nous calculons la taille du fichier pour la mettre en ko.
<td>{{ko}}ko</td>

Téléchargement des fichiers

src/Controller/FichierController.php

```
#[Route('/profil-telechargement-fichier/{id}', name: 'telechargement-fichier', requirements: ["id"=>"\d+"] )]
    public function telechargementFichier(int $id) {
        $doctrine = $this->getDoctrine();
        $repoFichier = $doctrine->getRepository(Fichier::class);
        $fichier = $repoFichier->find($id);
        if ($fichier == null){
            $this->redirectToRoute('ajout_fichier'); }
        else{
            return $this->file($this->getParameter('file_directory').'/'.$fichier->getNomServeur(), $fichier->getNomOriginal());
        }
    }
```

ajout-fichier.html.twig

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

{% block title %}
```

```
        {{parent()}}Ajouter un fichier
{% endblock %}

{% block body %}
        <div class="container-fluid">
                <h1 class="text-center text-primary mt-4 pt-4 display-1 fw-bold">
                        Ajouter un fichier</h1>
                <div class="row justify-content-center">
                        <div class="col-12 bg-white p-4 m-0 text-primary">
                                {{form(form)}}
                                </div>
                        </div>
                        <div class="row justify-content-center">
                                <div class="col-12 bg-white p-4 m-0 text-primary">
                                        <div class="table-responsive">
                                                <table class="table caption-top">
                                                        <caption>Liste des fichiers</caption>
                                                        <thead>
                                                                <tr>
                                                                        <th scope="col">Nom</th>
                                                                        <th scope="col">Date</th>
                                                                        <th
scope="col">Extension</th>
                                                                        <th scope="col">Taille</th>
                                                                        <th scope="col"></th>
                                                                </tr>
                                                        </thead>
                                                        <tbody>
                                                                {% for f in app.user.fichiers | sort((a, b)
=> a.nomOriginal <=> b.nomOriginal) %}
                                                                        <tr class="{{ cycle(['table-
primary', 'table-secondary'], loop.index0) }}">

        <td>{{f.nomOriginal}}</td>

                                                                                <td>{{f.dateEnvoi |
date("d-m-Y à H:i:s") }}</td>
                                                                                <td>
                                                                                        {% if f.extension
== 'pdf'%}
                                                                                                <i
class="bi bi-file-earmark-pdf"></i>
                                                                                        {% else %}
                                                                                                <i
class="bi bi-file-earmark-text"></i>
                                                                                        {% endif %}
                                                                                </td>
                                                                                {% set ko = f.taille /
1000 %}
                                                                                <td>{{ko}}
```

```
                                                                    ko</td>
                                                                    <td><a
href="{{path('telechargement-fichier', {'id':f.id})}}" target="_blank"><i class="bi bi-download
text-black"></i></td>
                                                                </tr>
                                                            {% endfor %}
                                                        </tbody>
                                                    </table>
                                                </div>
                                            </div>
                                        </div>
                                    </div>
                                {% endblock %}
```

Commentaire :
Nous pouvons maintenant télécharger un fichier en cliquant sur l'icône « download ». Pour cela, nous avons créé un lien qui va ouvrir un nouvel onglet. Nous appelons la route « telechargement-fichier » que nous avons mise en place précédemment. Nous précisons l'identifiant du fichier que nous souhaitons télécharger dans la fonction « path » => path('telechargement-fichier', {'id':f.id}
Le nom de la variable est ici « id » que nous mettons dans un tableau. Sa valeur est f.id qui contient l'identifiant du fichier dans la base de données.
Le fichier est récupéré dans le répertoire inaccessible directement par un navigateur, renommé avec son nom d'origine puis envoyé à l'utilisateur.