Gérer l’upload de fichiers

L’entité

- Avant de manipuler des fichiers, il faut penser modèle métier.
- Exemple : On veut enregistrer un produit avec une image.
- Champs de l'entité :
  - name : string
  - price : float
  - image : string, pour stocker le nom de l’image (pas le fichier lui-même !)
- ⚠️ **On ne stocke jamais le fichier en base**, uniquement le nom du fichier.

Configuration de Symfony pour gérer les fichiers

- On veut enregistrer les images dans un dossier spécifique.
- Déclaration du chemin dans config/services.yaml :

```yaml
parameters:
    uploads_directory: '%kernel.project_dir%/public/uploads/products’
```

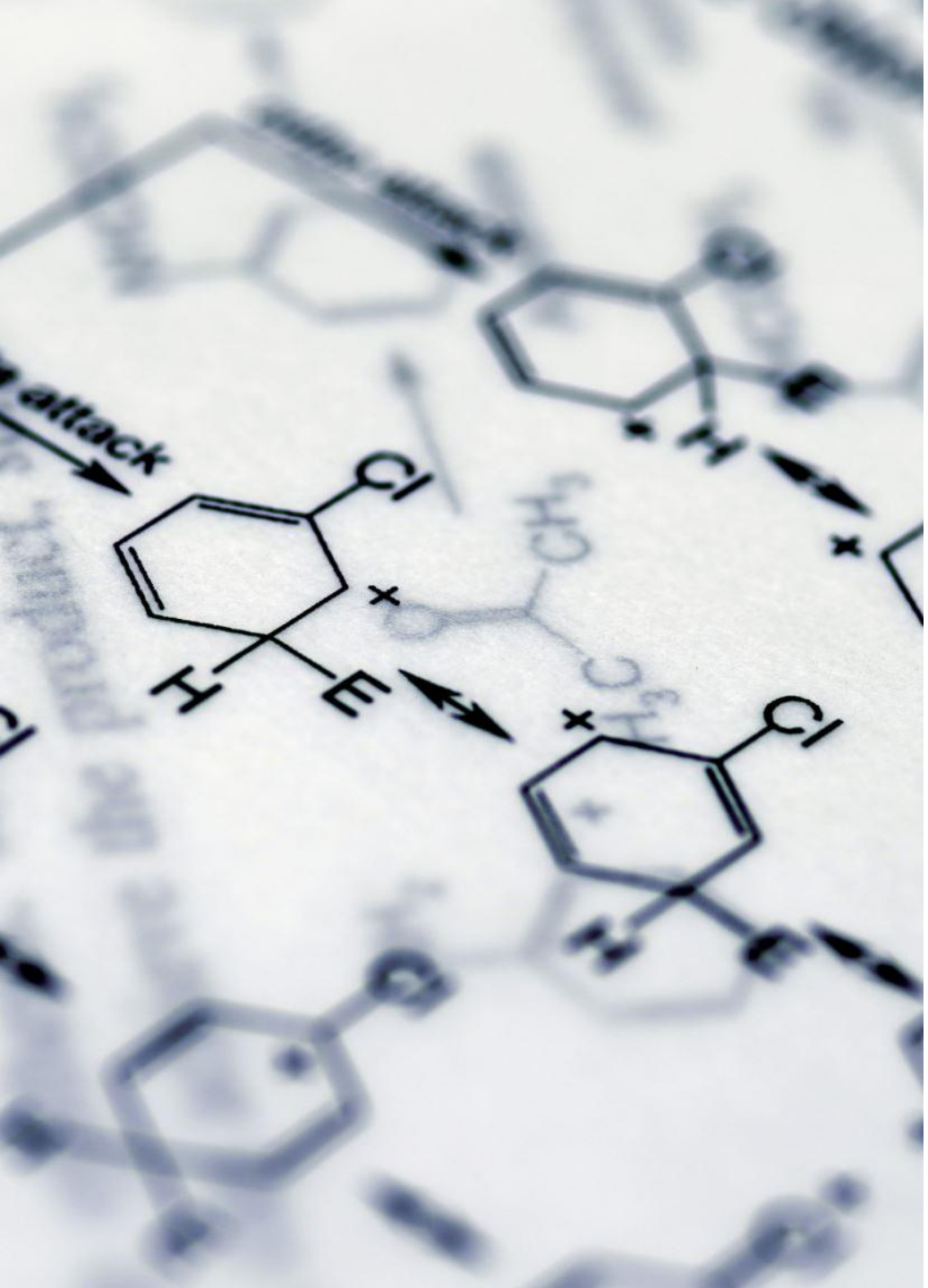
- Ce dossier doit être **accessible publiquement** pour permettre l'affichage dans les pages du site.

Création du formulaire

- Symfony gère très bien les fichiers via le type **FileType**.
- Dans le form builder :

```
use Symfony\Component\Form\Extension\Core\Type\FileType;
$builder
    ->add('name')
    ->add('price')
    ->add('image', FileType::class, [
        'label' => 'Image du produit',
        'mapped' => false, // Champ non lié à l'entité directement
    ])
;
```

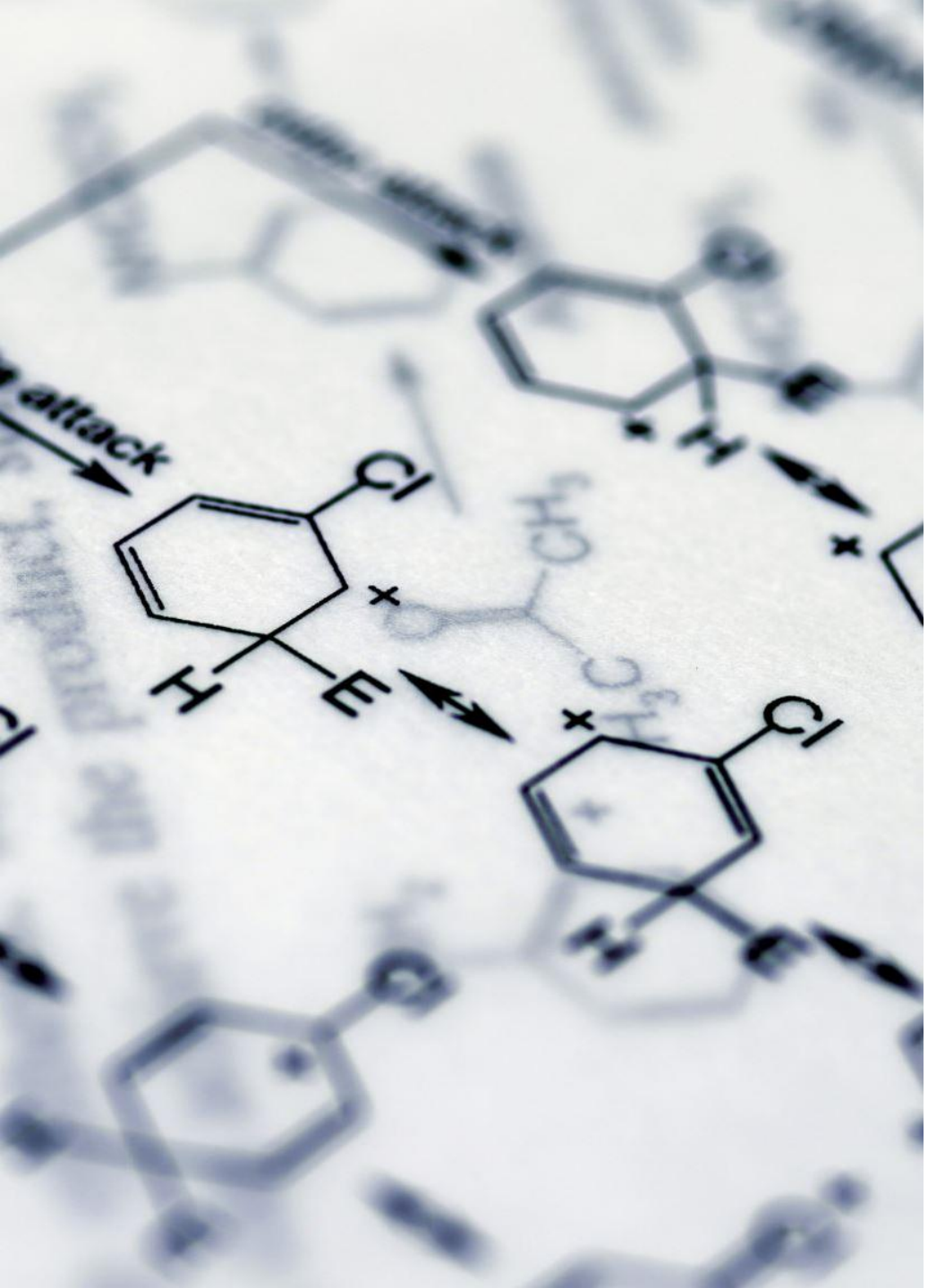
- mapped: false signifie que le champ n'est **pas automatiquement lié à une propriété de l'entité**. On le traitera nous-mêmes.



Traitement du fichier dans le contrôleur

- Une fois le formulaire soumis, on récupère l'objet UploadedFile et on déplace le fichier dans notre dossier.
- Extrait de contrôleur typique :

```
$imageFile = $form->get('image')->getData();
if ($imageFile) {
    $originalFilename = pathinfo($imageFile->getClientOriginalName(),
   PATHINFO_FILENAME);
    $safeFilename = $slugger->slug($originalFilename);
    $newFilename = $safeFilename.'-'.uniqid().'.'.$imageFile-
   >guessExtension();
    $imageFile->move(
        $this->getParameter('uploads_directory'),
        $newFilename
    );
    $product->setImage($newFilename);
}
```



Traitement du fichier dans le contrôleur

- Pourquoi ce traitement ?
  - On génère un nom sûr (éviter les caractères spéciaux)
  - On ajoute un identifiant unique
  - On déplace le fichier dans le répertoire public
  - On stocke le nom en base (dans image)

Affichage de l’image

- Dans une page Twig (ex : détail produit), on utilise asset() :

```
{% if product.image %}
    <img src="{{ asset('uploads/products/' ~ product.image) }}" alt="{{ product.name }}" width="200">
{% endif %}
```

- Le fichier est servi directement par le serveur web depuis /public.

Bonnes pratiques

Bonne pratique	Pourquoi
mapped: false	Gérer soi-même la logique de fichier
Nom unique via uniqid()	Évite les collisions de noms
Slugification du nom	Supprime les caractères spéciaux
Valider les types MIME	Sécurité contre les fichiers dangereux
Stockage dans public/	Permet l’accès depuis le navigateur
Ne pas stocker le fichier en base	Allège la base, meilleure performance
Supprimer l’ancien fichier	Lors d’une modification ou suppression

Pour aller plus loin

- Gérer l’édition : supprimer ou remplacer une ancienne image
- Upload multiple : pour une galerie de produits
- Intégration d’un CDN ou stockage externe (S3, Wasabi…)
- Affichage de miniatures avec LiipImagineBundle

Conclusion

- L’upload de fichiers avec Symfony est souple et puissant, à condition de respecter les bonnes pratiques :
  - bien séparer les responsabilités
  - sécuriser le traitement des fichiers
  - stocker uniquement les données nécessaires
- Grâce à cette approche, on obtient un code propre, maintenable et prêt pour la production.