# Transfert de fichiers avec HTTP

Saviez-vous qu'on peut téléverser des fichiers avec le protocole HTTP? On peut le faire par l'entremise des formulaires (<form>). Symfony propose un enrobage autour de ce mécanisme.

## FileType de Symfony

On a déjà rencontré les TextType, DatetimeType, SubmitType, etc : types de champ de formulaire offerts par Symfony. On a même créé nos propres formulaires pouvant s'imbriquer dans d'autres formulaires (par exemple AdresseType dans le formulaire du chomeur). Nous verrons maintenant le type **FileType**. C'est un type de champ permettant de téléverser (upload) des fichiers d'un ordinateur local vers le disque dur du serveur Web.

### Image du chômeur

ChomeQuiPeut veut offrir au chômeur la possibilité de téléverser une image de lui-même. Créons le formulaire qui rendra cela possible :

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

Ne donnez pas de nom de classe à associer au formulaire.

Modifiez le fichier **PhotoChomeurType.php** nouvellement généré comme suit :

```php
<?php
// ..\Form\PhotoChomeurType.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, SubmitType };

class PhotoChomeurType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
          ->add('imageChomeur', FileType::class)
          ->add('Envoyer', SubmitType::class);
    }
. . .
```

Remarquez le champ imageChomeur qui est de type **FileType**.

Créez manuellement la classe **PhotoChomeur** qui sera alimentée par le formulaire créé précédemment.

```php
<?php
// ..\src\Classe\PhotoChomeur.php
namespace App\Classe;

use Symfony\Component\HttpFoundation\File\UploadedFile;

class PhotoChomeur
{
   private $chomeurId;
   private $imageChomeur;
```

```
    public function getImageChomeur() : ?UploadedFile
    {
        return $this->imageChomeur;
    }
    public function setImageChomeur (UploadedFile $fichier = null)
    {
        $this->imageChomeur = $fichier;
    }
    public function getChomeurId()    { return $this->chomeurId; }
    public function setChomeurId($id) {$this->chomeurId= $id;}
}
```

C'est une simple classe PHP. Ce n'est pas une entité car on n'a pas besoin de sauvegarder ses instances en BD. L'attribut stratégique de cette classe est **$imageChomeur** qui est de type **UploadedFile.** J'ai mis en jaune ce qui concerne ce type. Nous avons également « typehinté » l'accesseur et le mutateur pour clairement indiquer les endroits où l'on se servira de ce type. Notez que les typehint sont facultatifs en php.

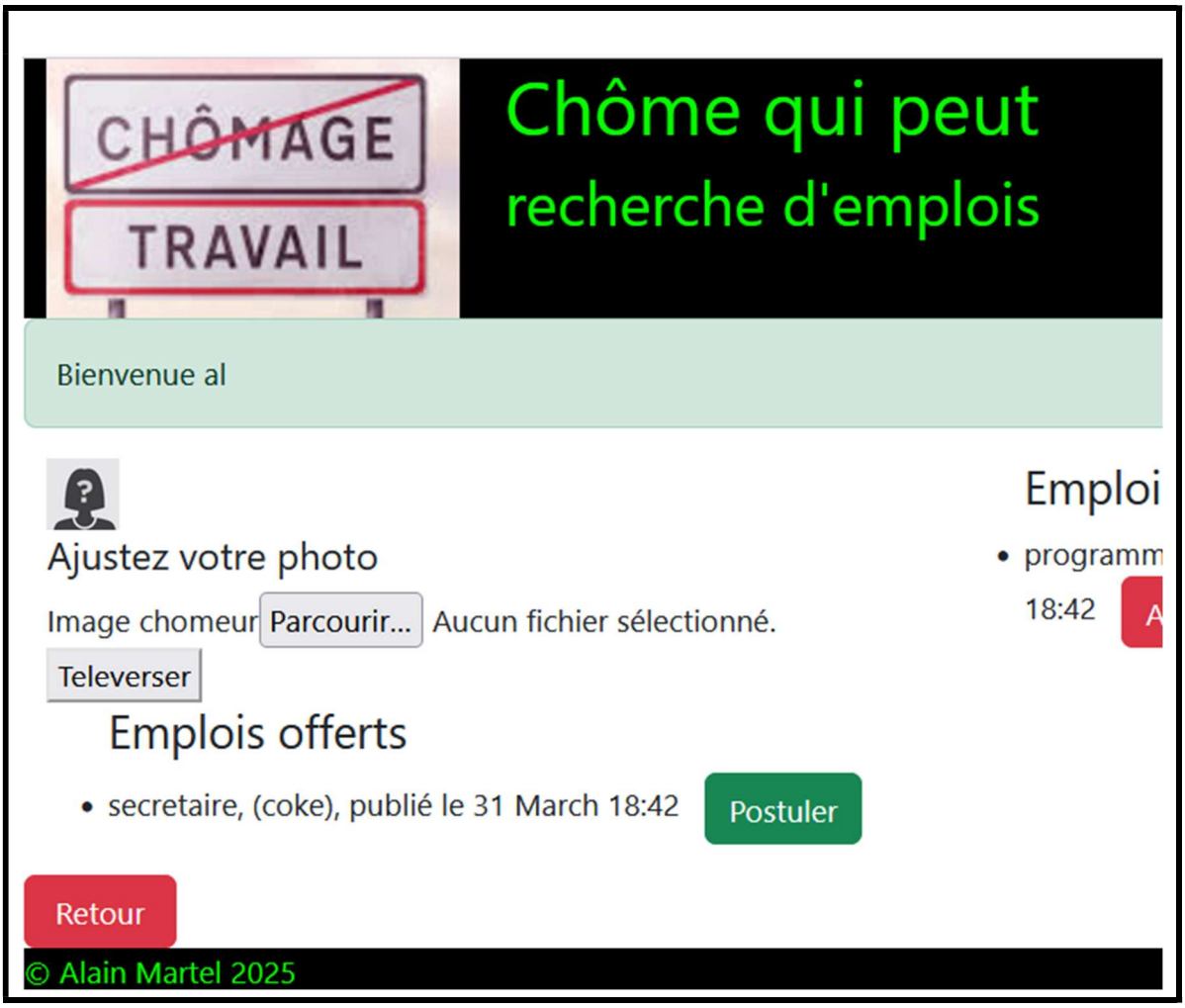
Modifier la route **accueilChomeur** pour y ajouter le traitement du fichier de photo :

```
// ChomeurController.php
. . .
use App\Form\PhotoChomeurType;
use App\Classe\PhotoChomeur;
.  . .
#[Route('/accueilChomeur', name:accueilChomeur')]
public function accueilChomeur(ManagerRegistry $doctrine,
                                Request $request) :Response
{
    . . .
    $ chomeur = $em
                ->getRepository(Chomeur::class)
                ->find($request->getSession()->get('chomeur_connecte')
                ->getId());

    $nomFichierImage =  __DIR__ .
                       '/../../public/images/chomeur' .
                       $chomeur->getId() . '.png';

    //Si l'image du chomeur n'existe pas on met l'image par défaut
    if (file_Exists($nomFichierImage))
        $image = 'images/chomeur' . $chomeur->getId() . '.png';
    else
        $image = 'images/pasImage.png';

    $photoChomeur = new PhotoChomeur;
    $photoChomeur->setChomeurId($chomeur->getId());
    // On créé le formulaire avec le PhotoChomeurType et on lui passe une
    // instance de la classe PhotoChomeur
    $formPhotoChomeur = $this->createForm(PhotoChomeurType::Class,
                                          $photoChomeur);

    $formPhotoChomeur->handleRequest($request);
    if ($formPhotoChomeur->isSubmitted())
    {
      if ($formPhotoChomeur->isValid())
      {
         $codeErreur = 0;
         if ($photoChomeur->televerse($codeErreur))
```

```
        {
           $this->addFlash('notice', 'image du chômeur téléversée avec
                                      succès!');
        }
        else
           $this->addFlash('erreur', "Erreur ($codeErreur) lors du
                                      téléversement de l'image");
     }
     else
     {
           $this->addFlash('erreur', "Formulaire invalide ");
     }
}
      . . .
      return $this->render('dossier_chomeur.html.twig',
                 ['tabOE_NonPostulees' => $tabOENonPostulees,
                  'tabOE_Postulees' => $chomeur->getPostulations(),
                  'tabEntrep' => $tabEntrep,
                  'filtreEntrepNom' => $filtreEntrepNom,
                  'image' => $image,
                  'formPhotoChomeur' => $formPhotoChomeur->createView() ]);

. . .
```

Dans le twig **accueilChomeur.html.twig**, affichez l'image du chômeur et le formulaire de téléversement:

```
 . . .
{# accueilChomeur.html.twig #}
. . .
   <section class="row">
      <h2>Dossier de {{app.session.get('chomeur_connecte').nom}}</h2>
      <img class="col-1" src='{{ asset(image) }}' alt='image non disponible'>
   </section>
. . .
     <article class="col-4">
       <h5>Ajustez votre photo</h5>
        {{ form(formPhotoChomeur)}}
     </article>
   . . .
```




Pour le téléversement (upload) il faut développer la méthode **televerse()** de la classe **PhotoChomeur**.:

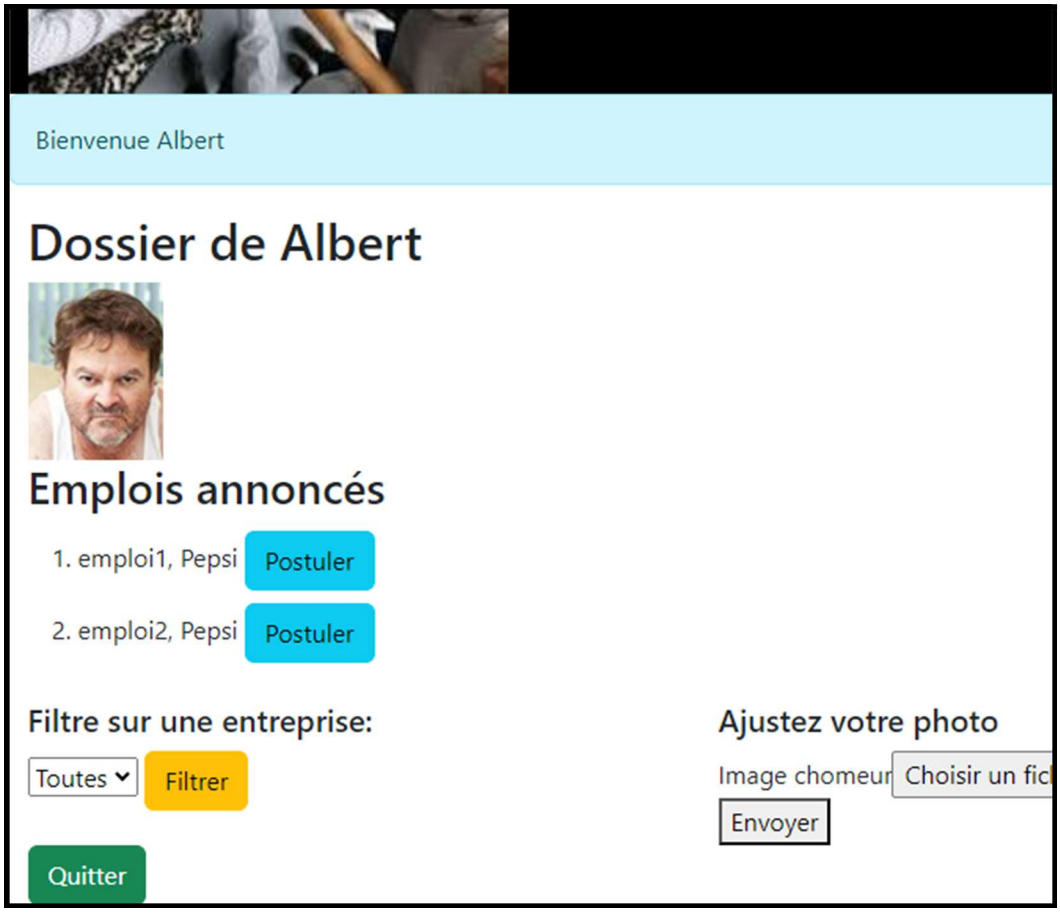
```
// ..\src\Classe\PhotoChomeur.php
. . .
// C'est une fonction booléenne : true = succès false = échec. Le
// param $codeErreur retournera un code d'erreur au contexte appelant cette
// méthode
public function televerse(&$codeErreur = 0)
{
   $codeErreur = 0;
    // on récupère l'extension du fichier à téléverser
    $type = $this->imageChomeur->getClientMimeType();
    // on accepte seulement les formats jpg, png et gif
    if ($type == 'image/gif' ||
        $type == 'image/png' ||
        $type == 'image/jpeg' )
    {
       // on construit le nom du dossier de destination
       $nomDossier = __DIR__ . '/../../public/images';
       // on construit le nom du fichier de destination
       $nomFichier = "chomeur$this->chomeurId.png";

       // on fait move le fichier reçu dans son dossier/fichier final
       $this-> imageChomeur ->move($nomDossier, $nomFichier);
```

```
            return true;
        }
        else
        {
            // -3 : mauvaise extension de fichier
            $codeErreur = -3;
            return false;
        }
    }
```

Remarquez les méthodes de la classe UploadedFile :

**->getClientMimeType()** : permet de récupérer l'extension du fichier à téléverser

**->move()** : Un téléversement se fait par défaut dans un dossier temporaire configuré dans le PHP.ini sous un nom aléatoire construit par le serveur. Le ->move() permet de déplacer ce fichier temporaire vers son dossier final avec un nom que notre code peut construire à sa guise.




## Traitement des erreurs

Des erreurs peuvent survenir lors du téléversement du fichier, par exemple on peut refuser certains types de fichiers. Les erreurs de niveau televerse() seront attrapées ainsi dans notre classe PhotoChomeur :

```
<?php
// ..\src\Classe\PhotoChomeur.php
namespace App\Classe;
. . .
  public function televerse(&$codeErreur = 0)
  {
     if ($this->imageChomeur->isValid())
     {
          // on récupère l'extension du fichier à téléverser
          $type = $this->imageChomeur->getClientMimeType();
. . .
     }
     else
     {
        // isValid() a détecté une erreur
        $codeErreur = $this->imageChomeur->getError();
        return false;
     }
}
```

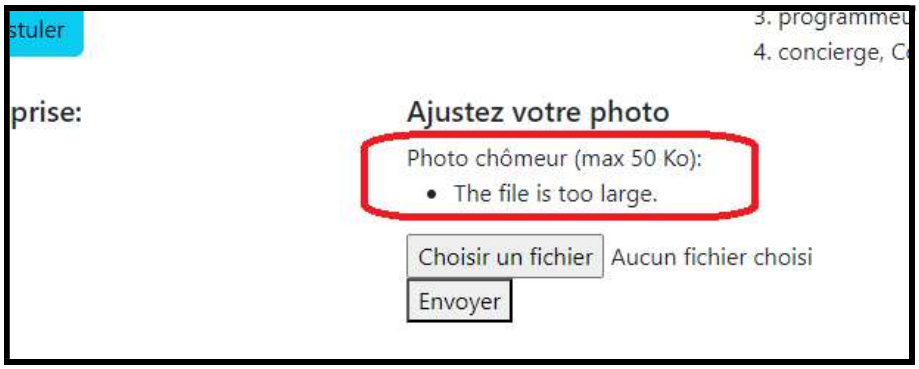
Les méthodes ->isValid() et ->getError() sont des méthodes de la classe UploadedFile et permettent de relayer les codes d'erreur au contexte appelant, c'est-à-dire au contrôleur.

Un deuxième niveau de traitement d'erreur, le niveau soumission du formulaire est très important. Il permet entre autres d'éviter des dénis de service (denial of service ou DOS) en limitant la taille permise des fichiers téléversés. Téléverser plusieurs giga-octets devient vite couteux en ressources et expose notre site. Pour imposer une taille limite on doit décomposer l'affichage du formulaire dans notre TWIG et **ajouter une balise cachée (hidden)** juste avant le form_widget du champ FileType. Cette balise sera de type 'Hidden' sa 'value' indiquera le nombre maximal d'octets permis pour le téléversement et son 'name' sera MAX_FILE_SIZE :

```
{# dossier_chomeur.html.twig #}
. . .
<h1>Ajouter la photo d'un chômeur</h1>

{{ form_start(formPhotoChomeur) }}
  {{ form_label(formPhotoChomeur.imageChomeur, "Photo chômeur (max 50 Ko):") }}
   {{ form_errors(formPhotoChomeur.imageChomeur) }}
   <input type='Hidden' value='50000' name='MAX_FILE_SIZE'>
  {{ form_widget(formPhotoChomeur.imageChomeur) }}
  {{ form_rest(formPhotoChomeur)  }}
{{ form_end(formPhotoChomeur) }}
<hr>
<a href='{{ path("menu") }}'>retour</a>
```

Donc deux niveaux d'erreurs qui seront traités dans le contrôleur.

Une erreur de dépassement de taille donnera le rendu suivant :


stuler
3. programmeu
4. concierge, C
prise:
Ajustez votre photo
Photo chômeur (max 50 Ko):
The file is too large.
Choisir un fichier
Aucun fichier choisi
Envoyer