Symfony 4
Les formulaires

AYMEN SELLAOUTI

Introduction

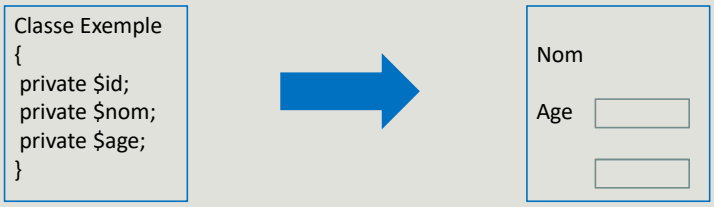
- Rôle très important dans le web
- Vitrine, interface entre les visiteurs du site web et le contenu du site
- Généralement traité en utilisant du html <form> … </form>
- Symfony et les formulaires : le composant Form
- Bibliothèque dédiée aux formulaires

Qu'est ce qu'un formulaire symfony

La philosophie de Symfony pour les formulaires est la suivante :

Vision 1

- Un formulaire est l'image d'un objet existant
- Le formulaire sert à alimenter cet objet.



Vision 2

- Un formulaire sert à récupérer des informations indépendantes de n'importe quel objet.

Comment créer un formulaire
Méthodes de création de formulaire

La création du formulaire se fait de 2 façons différentes :

1) Dans le contrôleur qui va utiliser le formulaire

2) En externalisant la définition dans un fichier

Comment créer un formulaire FormBuilder

La création d'un formulaire se fait à travers le Constructeur de formulaire FormBuilder

Exemple :

$monPremierFormulaire= $this->createFormBuilder($objetImage)

Pour indiquer les champs à ajouter au formulaire on utilise la méthode add du FromBuilder

La méthode add contient 3 paramètres :

1) le nom du champ dans le formulaire

2) le type du champ http://symfony.com/doc/current/reference/forms/types.html

3) un array qui contient des options spécifiques au type du champ

Exemple :

```
$monPremierFormulaire= $this->createFormBuilder($exemple)->add('nom' ,TextType::class)
                                                         ->add('age' , IntegerType::class)
```


Comment créer un formulaire Récupérer le formulaire avec getForm()

Pour récupérer le formulaire crée, il faut utiliser la méthode getForm()

Exemple :

```
$monPremierFormulaire= $this-createFormBuilder($exemple)->add('nom' ,TextType::class)
                                                        ->add('age' , IntegerType::class)
                                                        ->getForm();
```

Externalisation de la définition des formulaires Commande de génération de formulaire

Maker permet aussi d'automatiquement générer la classe du formulaire

php bin/console make:form FormNameType

Exemple :

php bin/console make:form PersonneType

Maker vous demandera si votre formulaire est associé à une entité ou non. Répondez selon votre besoin.

La récupération du formulaire au niveau des contrôleurs devient beaucoup plus facile :

$form = $this->createForm(TacheType::class, $entity);

Le second parameter n'est pas obligatoire

Externalisation de la définition des formulaires Commande de génération de formulaire

```php
<?php
namespace Rt4\AsBundle\Form;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class TacheType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('matache')
            ->add('date')
        ;
    }
```

Externalisation de la définition des formulaires AbstractType

Afin de rendre les formulaires réutilisables, Symfony permet l'externalisation des formulaires en des objets.

- Convention de nommage : L'objet du formulaire doit être nommé comme suit NomObjetType
- Cet objet doit obligatoirement étendre la classe AbstractType
- Deux méthodes doivent obligatoirement être implémentées :
  - buildForm(FormBuilderInterface $builder, array $options) qui est la méthode qui va permettre la création et la définition du formulaire
  - Il y a aussi la méthode configureOptions qui permet de définir l'objet associé au formulaire. Cette fonction est obligatoire si vous voulez associer votre form à une classe.

Affichage du formulaire dans TWIG CreateView

Afin d'afficher le formulaire crée, il faut transmettre la vue de ce formulaire à la page Twig qui doit l'accueillir.

La méthode createView de l'objet Form permet de créer cette vue

Il ne reste plus qu'à l'envoyer à la page twig en question

Exemple :

```
$form= $this->createForm (ExempleType::class,$exemple) );


return $this->render('Rt4AsBundle:Default:myform.html.twig', array(
                                                    'form'=>$form->createView()));
```

Affichage du formulaire dans TWIG form

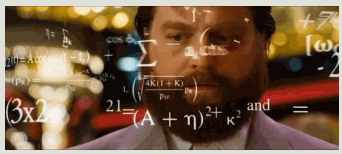
Deux méthodes permettent d'afficher le formulaire dans Twig :

1) Afficher directement la totalité du formulaire avec la méthode form

{{ form(nomDuFormulaire) }}

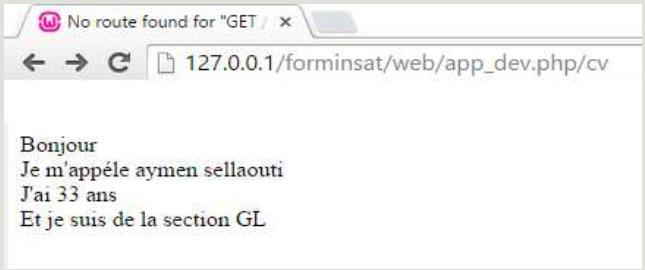
2) Afficher les composants du formulaire séparément un à un (généralement lorsqu'on veut personnaliser les différents champs)

Exercice



Créer un formulaire qui n'est associé à aucune classe.

Il devra contenir les champs nécessaires afin de faire en sorte que l'exercice du portfolio ne récupère plus les informations à travers la route mais à travers le formulaire :



Affichage du formulaire dans TWIG
Les composants du formulaire

form_start() affiche la balise d'ouverture du formulaire HTML, soit <form>. Il faut passer la variable du formulaire en premier argument, et les paramètres en deuxième argument. L'index attr des paramètres, et cela s'appliquera à toutes les fonctions suivantes, représente les attributs à ajouter à la balise générée, ici le <form>. Il nous permet d'appliquer une classe CSS au formulaire, ici form-horizontal.

Exemple : {{ form_start(form, {'attr': {'class': 'form-horizental'}}) }}

form_errors() affiche les erreurs attachées au champ donné en argument.

form_label() affiche le label HTML du champ donné en argument. Le deuxième argument est le contenu du label.

Affichage du formulaire dans TWIG (3)
Les composants du formulaire (2)

form_widget() affiche le champ HTML lui-même (que ce soit <input>, <select>, etc.).

Exemple : {{ form_widget(form.title, {'attr': {'class': 'form-control'}}) }}

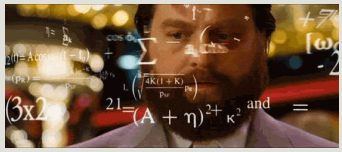
form_row() affiche le label, les erreurs et le champ.

form_rest() affiche tous les champs manquants du formulaire.

form_end() affiche la balise de fermeture du formulaire HTML, soit </form>.

Remarque : Certains types de champ ont des options d'affichage supplémentaires qui peuvent être passées au widget. Ces options sont documentées avec chaque type, mais l'option attr est commune à tous les types et vous permet de modifier les attributs d'un élément de formulaire.

Exercice



Reprenez le formulaire que vous avez crée et changer le en décortiquant chaque champs.

Passer une URL à l'objet du formulaire

Ne pouvons pas accéder dans la classe AbstractType à la méthode generateUrl afin de modifier l'action du formulaire, il faut donc procéder ainsi :

- Utiliser le troisième paramètre de la méthode **createForm.** C'est un tableau associatif contenant un ensemble d'option. On peut y ajouter deux clé :
  - **action** : pour ajouter l'url de l'action
  - **method** : si vous voulez modifier l'attribut method qui est par défaut à post.

```
$form = $this->createForm(FakeFormType::class, null ,array(
    'action' => $this->generateUrl('personne.add'),
    'method' => 'GET'
));
```

Gestion de la soumission des Formulaires

La gestion de la soumission des formulaires se fait à l'aide de la méthode handleRequest($request)

HandleRequest vérifie si la requête est de type POST. Si c'est le cas, elle va mapper les données du formulaire avec l'objet affecté au formulaire en utilisant les setters de cet objet.

Cette fonction prend en paramètre la requête HTTP de l'utilisateur qui est encapsulé dans Symfony au sein d'un objet de la classe Request de HttpFoundation que nous avons vu dans le Skill sur les controleur.

Pour récupérer cet objet dans une action on l'indiquera dans ses paramètres :

```
public function showFormAction(Request $request)
```

Exemple :

```
$form->handleRequest($request);

//On verifie si le formulaire a été soumis et s'il est valide
if ($form->isSubmitted() && $form->isValid()) {

  // ToDo

}
```

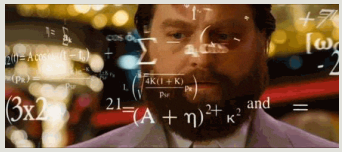

Récupérer les données envoyées

Afin de récupérer les données envoyées à votre formulaire vous avez deux méthodes.

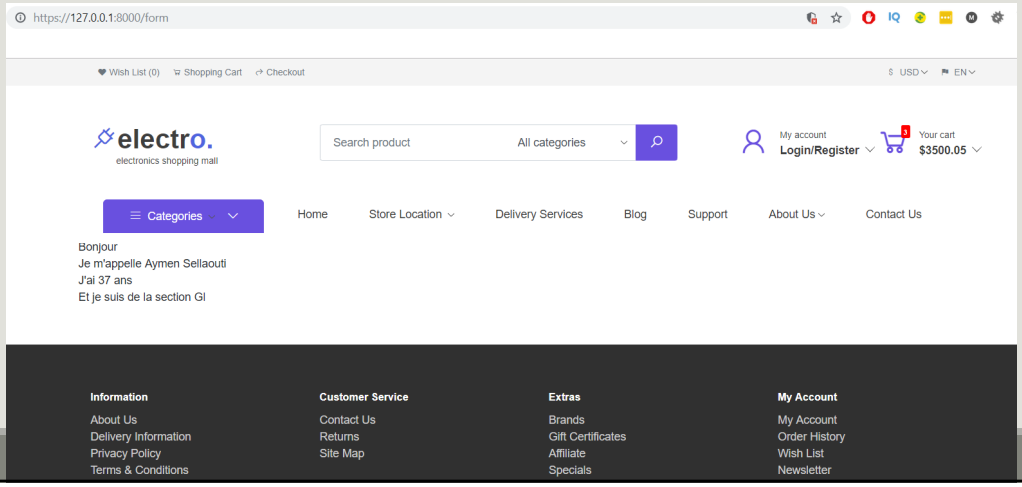
1 Récupérer les données avec la méthode getData() de l'objet form. Cette méthode vous retourne un tableau associatif contenant les données transmises à travers le formulaire. Chaque élément aura comme clé le contenu de l'attribut name dans le formulaire.

2 En passant comme deuxième paramètre à objet associé au formulaire, la fonction handleRequest irriguera automatiquement cet objet avec les données transmises à travers le formulaire.

Exercice



Récupérer les données envoyer à travers le formulaire et afficher le résultat.



Les propriétés d'un champ dans le formulaire

Le troisième paramètre de la méthode add est un tableau d'options pour les attributs du formulaire

Parmi les options communes à la majorité des champs nous citons :

- label : pour le label du champ si cette option n'est pas mentionné alors le label sera le nom du champ
- required : Permet de dire si le champ est obligatoire ou non (Par défaut l'option required est défini à true)

Les principaux types dans le formulaire
Liste des types

Les formulaires sont composés d'un ensemble de champs

Chaque champ possède un nom, un type et des options

http://symfony.com/doc/current/forms.html

Symfony propose une grande panoplie de types de champ

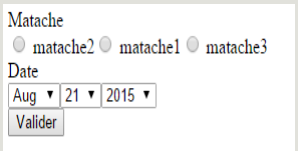
Texte	Choix	Date et temps	Divers	Multiple	Caché
TextType TextareaType EmailType IntegerType MoneyType NumberType PasswordType PercentType SearchType UrlType RangeType	ChoiceType EntityType CountryType LanguageType LocaleType TimezoneType CurrencyType	DateType DatetimeType TimeType BirthdayType	CheckboxType FileType RadioType	CollectionType RepeatedType	HiddenType CsrfType

Les principaux types dans le formulaire
Le type choice

Type spécifique aux champs optionnels (select, boutons radio, checkboxs)

Pour spécifier le type d'options qu'on veut avoir il faut utiliser le paramètre expanded. S'il est à false (valeur par défaut) alors nous aurons une liste déroulante. S'il est à true alors nous aurons des boutons radio ou des checkbox qui dépendra du paramètre multiple

Exemple :



Expanded=true Expanded=false

http://symfony.com/doc/current/reference/forms/types/choice.html

Les principaux types dans le formulaire (3) Le type Entity

Champ choice spécial

Les choices (les options) seront chargés à partir des éléments d'une entité Doctrine

```
->add('emploi',EntityType::class, array(
    'class' => 'Tekup\BdBundle\Entity\Emploi',
    'choice_label'=>'designation',
    'expanded'=>false,
    'multiple'=>true)
)
```

Balise HTML	expanded	multiple
Liste déroulante	false	false
Liste déroulante (avec attribut `multiple`)	false	true
Boutons radio	true	false
Cases à cocher	true	true

http://symfony.com/doc/current/reference/forms/types/entity.html

Personnaliser le choice label

Afin de personnaliser ce que vous voulez afficher dans vos choix, vous avez deux solutions :

1. Définir la méthode magique **to_string** de votre entité, c'est la méthode appelé par défaut en cas d'absence d'une information sur ce qu'il faut afficher.

2. Affecter à la propriété **choice_label** une **callback function** qui retournera la chaine à afficher

```
->add('formateur', EntityType::class, array(
    'choice_label' => function(Formateur $formateur) {
      return (
      sprintf(
            "%s-%s", $formateur->getName(),
                     $formateur->getField()
      )
      );
    }
)
```

EntityType query_builder

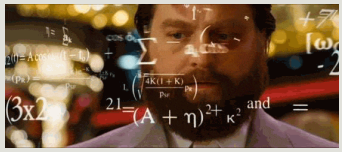
Afin de customiser la liste de choix de l'utilisateur vous pouvez utilisé la propriété **query_builder**

```
$builder->add('users', EntityType::class, [
'class' => User::class,
'query_builder' => function (EntityRepository $er) {
    return $er->createQueryBuilder('u')
        ->orderBy('u.username', 'ASC');
},
'choice_label' => 'username',
]);
```

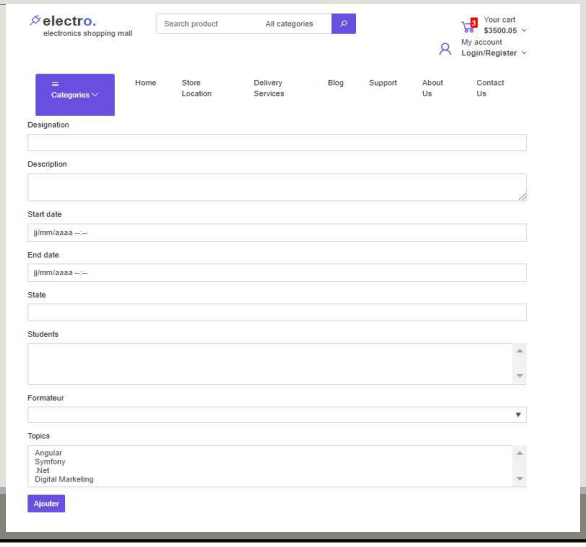
Vous pouvez aussi définir la liste de choix en surchargeant la propriété **choices** en injectant le repository de votre entité et en l'utilisant.

Appeler la fonction de votre repository qui retournera une collection des objets que vous voulez afficher.

Exercice



- Créer un contrôleur permettant de gérer vos formations.
- Créer l'action permettant d'afficher le formulaire de l'ajout d'une formation.
- Pensez à utiliser l'option widget de vos dates.



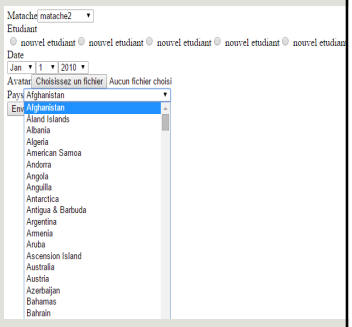
Les principaux types dans le formulaire (4) Le type country

Affiche la liste des pays du monde

La langue d'affichage est celle de la locale de votre application (config.yml)

Exemple

```
->add('pays',CountryType::class)
```



http://symfony.com/doc/current/reference/forms/types/country.html

Ne pas afficher un champs du formulaire

- Dans certains cas, vous n'avez pas envie d'afficher un champs de votre entité. Prenons l'exemple de l'état. Par défaut et lorsque vous créer une formation, vous voulez qu'elle soit active. Ce n'est pas un choix dépendant du créateur.
- Votre objet **form** contient une méthode **remove** (l'opposé de add) qui permet de supprimer un champs.
- Pensez à ajouter une valeur au champs supprimé ou bien ajouter une **valeur initiale** au niveau de l'entité à cet attribut.

Les principaux types dans le formulaire (4) Le type file

- Le type file permet l'upload de n'importe quel type de fichier.
- Créer un champ de ce type dans votre form et mettez l'option mapped à false.
- Le champ permet de récupérer un objet de type uploadedFile contenant le path de l'objet à uploader

Afin de récupérer ce champs utiliser votre objet form et accéder au paramètre ayant le même nom que votre propriété. Ensuite via la méthode getData récupérer votre objet. Exemple pour une propriété image : `$monImage = $form['image']->getData();`

- Pour pouvoir gérer cet objet il faut le copier dans le répertoire web de votre projet et de préférence dans un dossier spécifique pour vos images ou vos upload.
- Attribuer un nom unique à votre fichier pour ne pas avoir de problème lors de l'ajout de fichier ayant le même nom (vous pouvez utiliser la méthode suivante md5(uiniqueid());
- Pour récupérer l'extension vous pouvez utiliser la méthode guessExtension de votre objet file.
- Pour déplacer votre fichier utiliser la méthode move($pathsrc,$pathdest) de votre objet file
- __DIR__ vous donne le path de l'endroit ou vous l'utilisez.
- Vous pouvez créer un paramètre dans services.yml afin d'y stocker le path de votre dossier et le récupérer dans le Controller avec la méthode getParameter('nom du paramètre');
- Remarque : `%kernel.root_dir%` vous permet de récupérer le path du dossier app

http://symfony.com/doc/current/reference/forms/types/file.html

Customiser vos Form avec Bootstrap

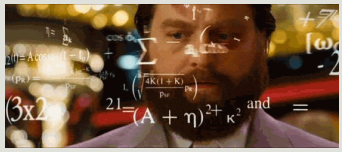
Afin d'intégrer directement **bootstrap** sur vos formulair, il suffit de :

- Spécifier à symfony dans le fichier **twig.yml** que vous voulez du Bootstrap pour vos formes.
- Informer la Twig qui contient vos formulaire

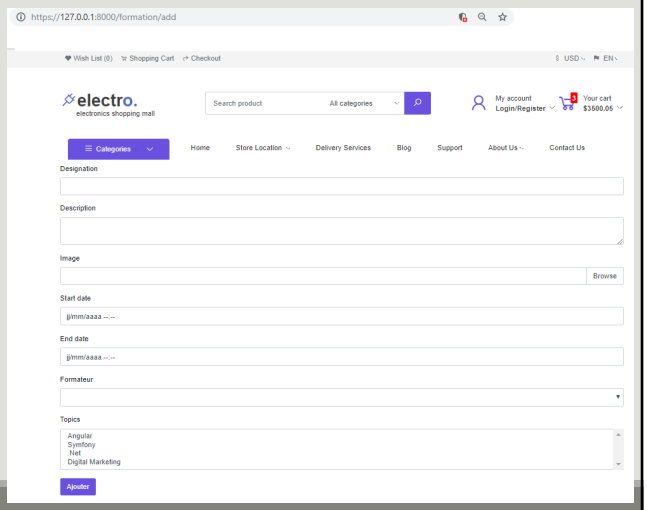
qu'elle doit utiliser ce thème la

```
twig:
    default_path: '%kernel.project_dir%/templates'
    debug: '%kernel.debug%'
    strict_variables: '%kernel.debug%'
    form_themes:['bootstrap_4_layout.html.twig']
```

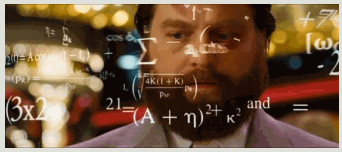
Exercice



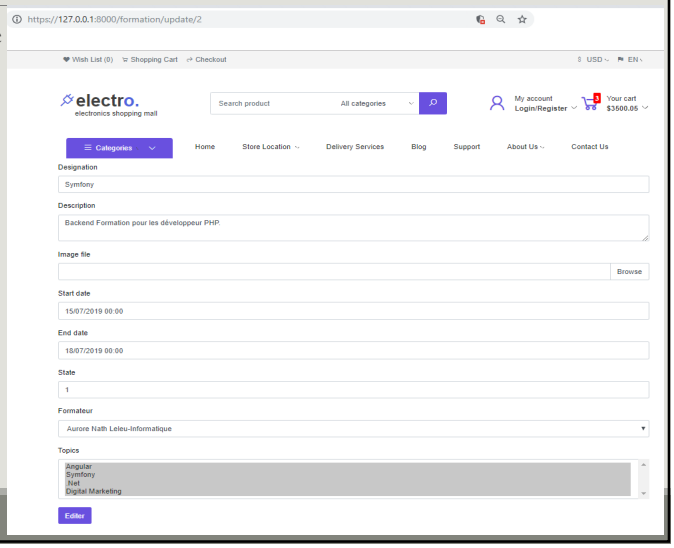
- Ajouter la partie gestion dans votre contrôleur afin d'ajouter une formation et de la mettre à jour.
- Enlever le champs state de l'affichage et mettez le à 1 par défaut.
- Enlevez le champs students
- Ajoutez un champs image et mettez en place le mécanisme d'upload de l'image.



Exercice



- Ajouter la fonctionnalité de mise à jour d'une formation.
- Enlevez le champs students



Les validateurs
Définition

Le validateur est conçu pour valider les objets selon des *contraintes*.

Le validateur de symfony est utilisé pour attribuer des contraintes sur les formulaires.

La validation peut être faite de plusieurs façons :

- YAML (dans le fichier validation.yml dans le dossier /Resources/config du Bundle en question)
- Annotations **sur l'entité de base du formulaire**
- XML
- PHP

La méthode isValide() du FORM déclenche le processus de validation

http://symfony.com/doc/current/reference/constraints.html

Exemple Validateur

```php
<?php
namespace AppBundle\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;
/**
 * @ORM\Table(name="personne")
*/
class Personne
{
    /**
     * @var int
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
    /**
     * @Assert\File(mimeTypes = {"application/pdf"})
     * @ORM\Column(name="path", type="string")
     */
// …
}
```

Ici nous indiquons à Symfony qu'il ne faut accepter que les fichiers dont le type est pdf

Les validateurs : Les annotations

Afin de pouvoir utiliser les annotations de validation il faut importer la class Constraints

use Symfony\Component\Validator\Constraints as Assert;

Syntaxe :

@Assert\MaContrainte(option1="valeur1", option2="valeur2", …)

Exemples :

@Assert\NotBlank(message = " Ce champ ne doit pas être vide ")

@Assert\Length(min=4, message="Le login doit contenir au moins {{ limit }} caractères.")

@Assert\Url()

https://symfony.com/doc/3.4/validation.html

Enlever la validation HTML

Afin d'enlever la validation html ajouter le mot clé novalidate à votre form

```
{{ form_start(form, {'attr': {'novalidate': 'novalidate'}}) }}
{{ form_widget(form) }}
{{ form_end(form) }}
```

Les annotations : Les contraintes de base

Contrainte	Rôle	Options
NotBlank Blank	La contrainte NotBlank vérifie que la valeur soumise n'est ni une chaîne de caractères vide, ni NULL. La contrainte Blank fait l'inverse.	-
True False	La contrainte True vérifie que la valeur vaut true, 1 ou "1". La contrainte False vérifie que la valeur vaut false, 0 ou "0".	-
NotNull Null	La contrainte NotNull vérifie que la valeur est strictement différente de null.	-
Type	La contrainte Type vérifie que la valeur est bien du type donné en argument.	type (option par défaut) : le type duquel doit être la valeur, parmi array, bool,int, object

Les annotations : Nombre, date

Contrainte	Rôle	Options
Range	La contrainte Range vérifie que la valeur ne dépasse pas X, ou qu'elle dépasse Y.	min : nbre de car minimum max : nbre de car maximum minMessage : msg erreur nbre de car min maxMessage : msg erreur nbre de car max invalidMessage : msg erreur si non nmbre
Date	vérifie que la valeur est un objet de type Datetime, ou une chaîne de type YYYY-MM-DD.	-
Time	vérifie qque c'est un objet de type Datetime, ou une chaîne type HH:MM:SS.	-
DateTime	vérifie que c'est un objet de typeDatetime, ou une chaîne de caractères du type YYYY-MM-DD HH:MM:SS.	

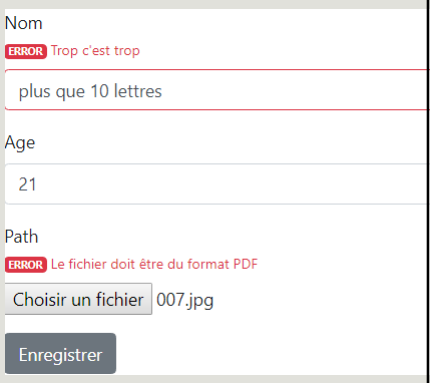
Les annotations : File

Contrainte	Rôle	Options
File	La contrainte File vérifie que la valeur est un fichier valide, c'est-à-dire soit une chaîne de caractères qui pointe vers un fichier existant, soit une instance de la classe File (ce qui inclut UploadedFile).	maxSize : la taille maximale du fichier. Exemple : 1M ou 1k. mimeTypes : mimeType(s) que le fichier doit avoir.
Image	La contrainte Image vérifie que la valeur est valide selon la contrainte précédente File (dont elle hérite les options), sauf que les mimeTypes acceptés sont automatiquement définis comme ceux de fichiers images. Il est également possible de mettre des contraintes sur la hauteur max ou la largeur max de l'image.	maxSize : la taille maximale du fichier. Exemple : 1M ou 1k. minWidth /maxWidth : la largeur minimale et maximale que doit respecter l'image. minHeight /maxHeight : la hauteur minimale et maximale que doit respecter l'image.

Validation Exemple

```
/**
 * @var string
 * @Assert\Length(min="3",max="10",maxMessage="Trop c'est trop")
 * @ORM\Column(name="nom", type="string", length=50)
 */
private $nom;

/**
 * @var string
 * @Assert\File(mimeTypes = {"application/pdf"},mimeTypesMessage="Le fichier doit être du format PDF")
 * @ORM\Column(name="path", type="string")
 */
private $path;
```

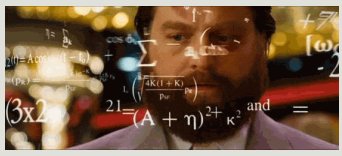


Valider des champs non mapés

Lorsque le champs que vous voulez valider est non mapé et que vous souhaiter le valider, il faut ajouter un paramètre constraints dans le tableau d'option de votre méthode add.

```
->add('imageFile', FileType::class, array(
    'mapped'=> false,
    'constraints'=> array(
                        new Image(),
                    );
))
```


Exercice



- Ajouter les validateurs nécessaires à votre formulaire. L'image doit être obligatoire dans le cas de l'ajout non dans le cas de la mise à jour.
- La date de fin doit être supérieur à la date de début de la formation.
- Forcer les champs que vous pensez required.
- L'upload ne doit concerner que les images d'une taille <3Mo