

# Symfony

# 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.

```
Classe Exemple  
{  
    private $id;  
    private $nom;  
    private $age;  
}
```



Nom   
Age

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`

# Comment créer un formulaire

## FormBuilder

---

La méthode `add` contient 3 paramètres :

- 1) le nom du champ dans le formulaire
- 2) le type du champ
- 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

## 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.



# 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
```

```
symfony console make:form FormNameType
```

- Exemple :

```
symfony console make:form PersonneType
```

# Externalisation de la définition des formulaires

## Commande de génération de formulaire

---

```
<?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

## Commande de génération de formulaire

---

- 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

# 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)

# 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'  
  form_themes: ['bootstrap_5_layout.html.twig']
```

# Récupérer les données envoyées

---

- 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. Si aucun objet n'est mappé, vous pouvez récupérer directement ces données.
- 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`.

# Récupérer les données envoyées

- Vous pouvez récupérer les données envoyées via votre formulaire en accédant au champ via la méthode `getData()` de l'objet form.

Exemple : `$form->getData()` retourne un tableau associatif avec les données envoyées par le formulaire.

- Chaque élément aura comme clé le contenu de l'attribut `name` dans le formulaire.

```
public function showFormAction(Request $request) {  
    $form->handleRequest($request);  
    if ($form->isSubmitted() ){  
        $data = $form->getData();  
        // ToDo  
    }  
}
```



# Exercice

---

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



# 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-horizontal'}}) }}`
- `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

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

**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 créé et changez-le en décortiquant chaque champ.



# 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'  
));
```

# 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
- Symfony propose une grande panoplie de types de champ

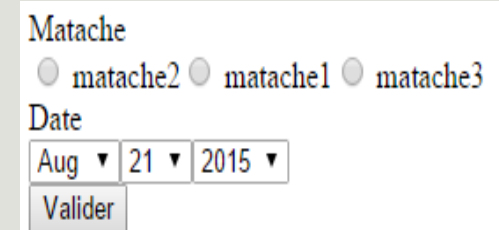
Texte	Choix	Date et temps	Divers	Multiple	Caché
TextType TextareaType EmailType IntegerType MoneyType NumberType PasswordType PercentType SearchType RangeType...	ChoiceType EntityType CountryType LanguageType LocaleType TimezoneType CurrencyType	DateType DatetimeType TimeType BirthdayType	CheckboxType FileType RadioType	CollectionType RepeatedType	HiddenType CsrfType
<a href="http://symfony.com/doc/current/forms.html">http://symfony.com/doc/current/forms.html</a>					

# Les principaux types dans le formulaire

## Le type choice

- Type spécifique aux champs optionnels (select, boutons radio, checkboxes)
- 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 :

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



A form snippet showing a choice type. It has a label "Matache" followed by three radio buttons labeled "matache2", "matache1", and "matache3". Below this is a "Date" field with three dropdowns for month ("Aug"), day ("21"), and year ("2015"). At the bottom is a "Valider" button.

Expanded=true



A form snippet showing a choice type. It has a label "Matache" followed by a dropdown menu. The dropdown is open, showing three options: "matache2" (which is highlighted), "matache1", and "matache3". Below this is a "Date" field with three dropdowns for month ("Aug"), day ("21"), and year ("2015"). At the bottom is a "Valider" button.

Expanded=false



# Les principaux types dans le formulaire

## 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 <code>multiple</code> )	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 chaîne à afficher pour chaque enregistrement. Elle prendra en paramètre l'instance de l'entité à traiter.

```
->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 (EntityManager $em) {  
        return $em->createQueryBuilder('u')  
            ->orderBy('u.username', 'ASC');  
    },  
    'choice_label' => 'username',  
]);
```

# Exercice

---

- Créer une méthode permettant d'ajouter une personne à travers le formulaire.

# Les principaux types dans le formulaire (4)

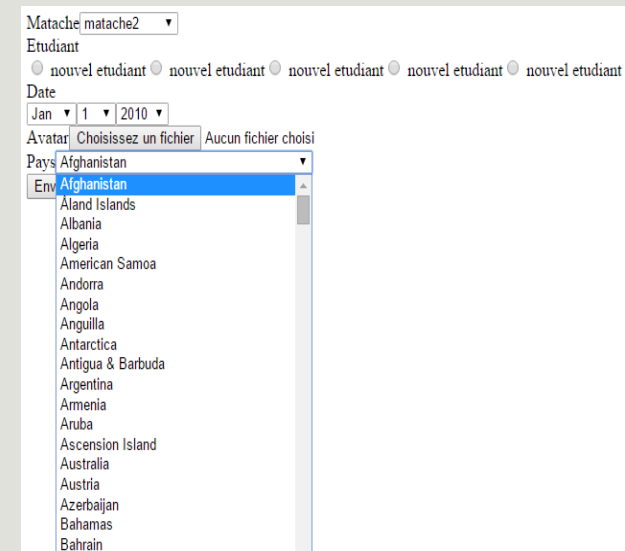
## Le type country

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)
```

A screenshot of a web form. At the top, there's a dropdown menu labeled 'Matache matache2'. Below it, a row of radio buttons is labeled 'Etudiant'. Then, a date picker is set to 'Jan 1 2010'. Below the date, there's an 'Avatar' label and a file upload button 'Choisissez un fichier' with the text 'Aucun fichier choisi'. The main part of the form is a 'Pays' (Country) dropdown menu. The dropdown is open, showing a list of countries starting with 'A': Aaland Islands, Albania, Algeria, American Samoa, Andorra, Angola, Anguilla, Antarctica, Antigua & Barbuda, Argentina, Armenia, Aruba, Ascension Island, Australia, Austria, Azerbaijan, Bahamas, and Bahrain. 'Afghanistan' is currently selected and highlighted in blue.

<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.

# Les principaux types dans le formulaire (4)

## Le type file

---

- 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(uniqueid())`);
- Pour récupérer le nom de votre file utiliser `getClientOriginalName()`
- 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 où 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`



# Le type file

```
/** @var UploadedFile $file */
$file = $form->get('file')->getData();
if ($file) {
    $originalFilename = pathinfo($file->getClientOriginalName(), PATHINFO_FILENAME);
    // this is needed to safely include the file name as part of the URL
    $safeFilename = $slugger->slug($originalFilename);
    $newFilename = $safeFilename . '-' . uniqid() . '.' . $file->guessExtension();
    // Move the file to the directory where brochures are stored
    try {
        $file->move(
            $this->getParameter('upload_directory'),
            $newFilename
        );
    } catch (FileNotFoundException $e) {
        // ... handle exception if something happens during file upload
    }
}
```

parameters:

upload\_directory: '%kernel.project\_dir%/public/uploads'

# Exercice

➤ Ajoutez un champs image pour l'entité Personne et mettez en place le mécanisme d'upload de l'image.



https://127.0.0.1:8000/formation/add

Wish List (0) Shopping Cart Checkout USD EN

**electro.**  
electronics shopping mall

Search product All categories

My account Login/Register Your cart \$3500.05

Categories Home Store Location Delivery Services Blog Support About Us Contact Us

Designation

Description

Image Browse

Start date  
jj/mm/aaaa --:--

End date  
jj/mm/aaaa --:--

Formateur

Topics  
Angular  
Symfony  
.Net  
Digital Marketing

Ajouter

# Exercice

---

- Ajouter la fonctionnalité de mise à jour d'une personne.



# 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

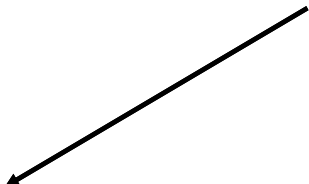
La méthode `isValid()` du FORM déclenche le processus de validation

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

# Exemple Valideur

```
<?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()
```

# 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".	-
<a href="#">NotNull</a> 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 nombre
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;
```

Nom

**ERROR** Trop c'est trop

plus que 10 lettres

Age

21

Path

**ERROR** Le fichier doit être du format PDF

Choisir un fichier 007.jpg

Enregistrer

# Valider des champs non mapés

---

Lorsque le champs que vous voulez valider est non mapé et que vous souhaitez 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.

