

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 symfony2

La philosophie de Symfony pour les formulaires est la suivante :

- 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

Comment créer un formulaire (1)

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 `FormBuilder`

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 (2)

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();
```

Comment créer un formulaire (3)

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

Affichage du formulaire dans TWIG(1)

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= $this->createFormBuilder($exemple) )->add('nom', TextType::class)
                                ->add('age', IntegerType::class)
                                ->add('save', ButtonType::class, array(
                                    'attr' => array('class' => 'save')
                                ));
                                ->getForm();

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

Affichage du formulaire dans TWIG (2)

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)

Affichage du formulaire dans TWIG (3)

Les composants du formulaire (1)

`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 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.

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 contrôleurs.

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

}
```

Externalisation de la définition des formulaires (1)

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
 - `getBlockPrefix()` qui retourne un identifiant unique pour le formulaire. Par convention c'est le nom du bundle et le nom de l'entité séparé avec des '_' et sans majuscule
 - Il y a aussi la méthode `setDefaultOptions` qui est facultative et qui permet de définir l'objet associé au formulaire

Externalisation de la définition des formulaires (2)

Doctrine permet d'automatiquement générer la classe du formulaire spécifique à une entité en utilisant la commande suivante :

```
php bin/console doctrine:generate:form BundleName:EntiyName
```

Exemple :

```
php bin/console doctrine:generate:form Rt4AsBundle:Tache
```

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

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

Externalisation de la définition des formulaires (3)

```
<?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 (4)

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

- Générer l'url au niveau du controleur
- Passer l'url dans le troisième paramètre optionnel de la méthode `createForm`
- Récupérer l'url dans l'objet du formulaire dans le tableau `$options` via son `label`

```
$form=$this->createForm  
(TacheType::class,$tache, array(  
    'action'=>$this->generateUrl  
( 'url_route' )  
));
```

```
public function buildForm(FormBuilderInterface $builder, array $options)  
{  
    $builder  
        ->setAction($options['action'])  
        ->setMethod('POST')  
        ->add('matache')  
        ->add('date')  
;  
}
```

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 (1)

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

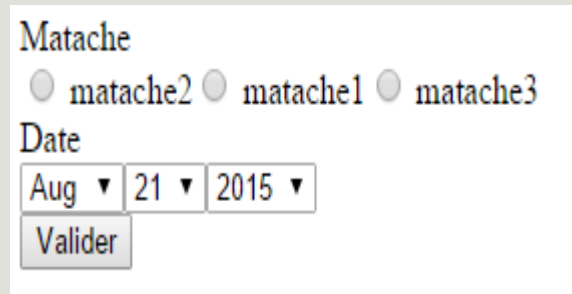
Les principaux types dans le formulaire (2)

Le type choice

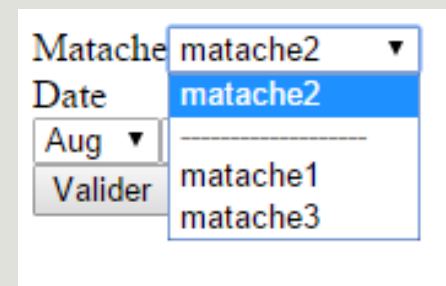
Type spécifique aux champs optionnels (select, boutons radio, checkboxes)

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 :

A form titled "Matache" with three radio buttons labeled "matache2", "matache1", and "matache3". Below the radio buttons is a "Date" section with three dropdown menus showing "Aug", "21", and "2015". At the bottom is a "Valider" button.

Expanded=true

A form titled "Matache" with a dropdown menu showing "matache2". Below it is a "Date" section with a dropdown menu showing "Aug" and a "Valider" button. The dropdown menu for "Matache" is open, showing a list of options: "matache2", "matache1", and "matache3".

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 <code>multiple</code>)	false	true
Boutons radio	true	false
Cases à cocher	true	true

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

Les principaux types dans le formulaire (4)

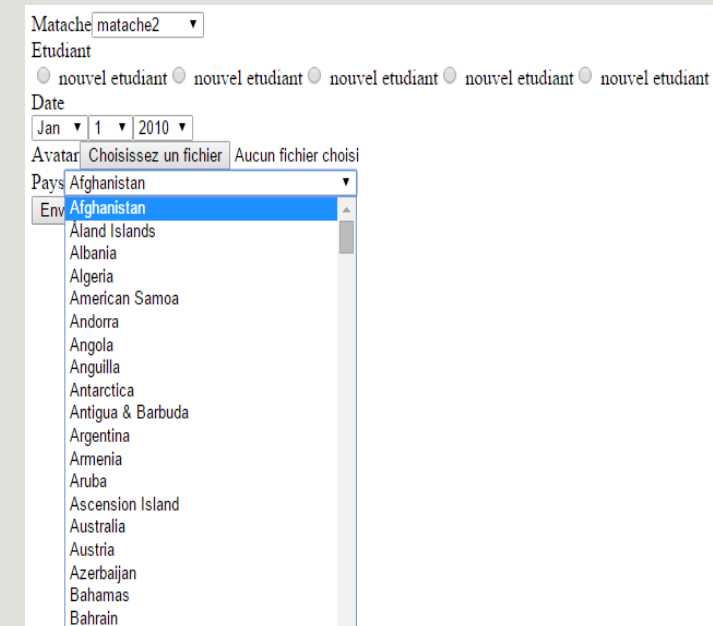
Le type country

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)
```



The screenshot shows a web form with several fields. At the top, there is a dropdown menu labeled 'Matache' with 'matache2' selected. Below it is a radio button group for 'Etudiant' with five 'nouvel etudiant' options. The 'Date' field shows 'Jan', '1', and '2010'. The 'Avatar' field has a button 'Choisissez un fichier' and the text 'Aucun fichier choisi'. The 'Pays' field is a dropdown menu with 'Afghanistan' selected. Below the 'Pays' field is an 'Env' field. The dropdown menu for 'Pays' is open, showing a list of countries: Afghanistan, Aland Islands, Albania, Algeria, American Samoa, Andorra, Angola, Anguilla, Antarctica, Antigua & Barbuda, Argentina, Armenia, Aruba, Ascension Island, Australia, Austria, Azerbaijan, Bahamas, and Bahrain.

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

Les principaux types dans le formulaire (4)

Le type file

- Le type `file` permet l'upload de n'importe quel type de fichier
- Le champ permet de récupérer un `objet` de type `file` contenant le `path` de l'objet à uploader
- 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(uniqueid())`);
- 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 `config.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 **config.yml** que vous voulez du Bootstrap pour vos forms dans la partie twig configuration.

➤ Informer la Twig qui contient vos formulaire qu'elle doit utiliser ce thème la

```
{% form_theme form 'bootstrap_3_layout.html.twig' %}
```

A partir de la version 3.4 de Symfony, Bootstrap a été intégré pour la gestion des formulaires. Vous n'avez qu'à ajouter le code suivant :

```
twig:  
  debug:                "%kernel.debug%"  
  strict_variables:     "%kernel.debug%"  
  form_themes:  
    - 'bootstrap_3_layout.html.twig'
```

```
twig:  
  form_themes: [ 'bootstrap_4_layout.html.twig' ]
```

Les validateurs

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

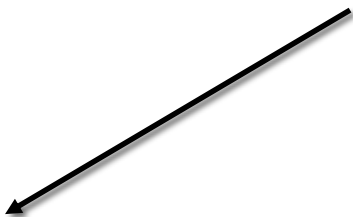
La méthode `isValid()` du FORM déclenche le processus de validation

<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()
```

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

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 qqe c'est un objet de type Datetime, ou une chaîne type HH:MM:SS.	-
DateTime	vérifie que c'est un objet de type Datetime, 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