FormType : la gestion des formulaires

lien vers la documentation

Forms (Symfony Docs)

Screencast Do you prefer video tutorials? Check out the Symfony Forms screencast series. Creating and processing HTML forms is hard and repetitive. You need to deal with

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

Vous commencez a comprendre le truc, symfony dispose de sa propre facon de gérer les formulaires, ceux ci doivent etre décrits dans un fichier de type FormType une classe qui (je vous le donne en mille) étend d'une classe générique AbstractType

Pourquoi utiliser le FormType :

- Séparation de la logique métier et de la présentation.
- Validation des données entrantes.
- Facilité de création de formulaires complexes.
- Intégration avec d'autres composants Symfony (Doctrine, Twig).

Créer un formulaire de base

Dans le dossier FormType créez un nouveau fichier par exemple ici ContactType pour un formulaire de contact.

Cette classe doit etendre de la classe AbstractType et doit implémenter 2 fonctions :

- buildForm(FromBuiderInterface $builder, array $options) dont le rôle est de construire la vue HTML du formulaire (labels, champs et messages

d’erreurs)

- la fonction configureOptions(OptionsResolver $options) : sert à configurer les options globales du formulaire. Ces options peuvent influencer le comportement du formulaire dans son ensemble, comme par exemple :
  - **Les options de données: Définir l'objet qui sera utilisé pour remplir les champs du formulaire (par exemple, une entité Doctrine).**
  - **Les options de validation:** Configurer des groupes de validation spécifiques.

Grace a la variable $builder vous allez pouvoir utiliser la methode add du FormBuilderInterface pour ajouter les champs de votre formulaire, le label , les différentes contraintes de validation, les classes css sur les input etc... comme dans l’exemple suivant pour le formulaire de contact

```
namespace App\FormType;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ContactType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builde
r, array $options)
    {
        $builder
            ->add('name', TextType::class, [
                'label' => 'Nom',
                'attr' => [
                    'class' => 'form-control',
                ],
                'constraints' => [
                    new Assert\NotBlank([
                        'message' => 'Veuillez saisir votre
```

```
nom.',
                    ]),
                    new Assert\Length([
                        'min' => 2,
                        'max' => 50,
                        'minMessage' => 'Votre nom doit con
tenir au moins {{ limit }} caractères.',
                        'maxMessage' => 'Votre nom ne peut
pas dépasser {{ limit }} caractères.',
                    ]),
                ],
            ])
            ->add('email', EmailType::class, [
                'label' => 'Email',
                'attr' => [
                    'class' => 'form-control',
                ],
                'constraints' => [
                    new Assert\NotBlank(),
                    new Assert\Email([
                        'message' => 'Veuillez saisir une a
dresse email valide.',
                    ]),
                ],
            ])
            ->add('message', TextareaType::class, [
                'label' => 'Message',
                'attr' => [
                    'class' => 'form-control',
                ],
                'constraints' => [
                    new Assert\NotBlank(),
                    new Assert\Length([
                        'min' => 10,
                        'minMessage' => 'Votre message doit
contenir au moins {{ limit }} caractères.',
                    ]),
                ],
```

```
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resol
ver)
    {
        $resolver->setDefaults([
            'data_class' => Contact::class, // Si vous liez
le formulaire à une entité
            'csrf_protection' => true,
        ]);
    }
}
```

Comme vous le voyez symfony gère aussi tout seul (ou presque) les **contraintes de validation** des données qu'il faut lister pour chaque champs a la clé **"constraints"** dans le tableau d'options de la **méthode add();**

Ces assertions vous permettent aussi de gérer ici les différents messages d'erreurs associés

plus d'infos sur les contraintes de validation : https://symfony.com/doc/current/validation.html#constraints

Prévoir le Controller et la méthode pour l'affichage du formulaire

```
use Symfony\Component\HttpFoundation\Request;

// ...

public function contactAction(Request $request)
{
    $form = $this->createForm(ContactType::class);

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

```
    if ($form->isSubmitted() && $form->isValid()) {
        // Le formulaire est valide, on peut traiter les do
nnées
        $data = $form->getData();
        // ... ici on affectuerais une action en base (ajou
t / modif) ou un envoi
        // de mail

        // puis une redirection vers une autre page
        $this->redirectToRoute("route_name")
    }
    // sinon : si on arrive en methode GET ou si le formula
ire n'est pas valide
    //on le réaffiche
    return $this->render('contact.html.twig', [
        'form' => $form
    ]);
}
```

Construire le formulaire dans le Template

2 solutions : On laisse Symfony se débrouiller

```
{# templates/task/new.html.twig #}
{{ form_start(form, {'attr': {'novalidate': 'novalidate'}})
}}
    {{ form_widget(form) }}
{{ form_end(form) }}
```

On place les champs un peu comme on veut “a la main” (plus facile pour la mise en forme)

```
{% form_start(form) %}

    {{ form_row(form.name) }}

    {{ form_row(form.email) }}
```

```
    {{ form_row(form.subject) }}

    {{ form_row(form.message) }}

    <button type="submit">Envoyer</button>

{% form_end(form) %}
```

⇒ pour l'affichage des erreurs ,utiliser la fonction form_errors() que vous pouvez positionner la ou vous voulez si vous avez décomposé le fomulaire en form_row()

```
{# render only the error messages related to this field #}
{{ form_row(form.name) }}
{{ form_errors(form.name) }}

{# render any "global" errors not associated to any form fiel
{{ form_errors(form) }}
```

Pour plus d'informations sur la customisation de l'affichage d'un formulaire TWIG rendez vous dans la documentation ici :

https://symfony.com/doc/current/form/form_customization.html

Utiliser les form_themes

pour rajouter des classes automatiquement sur les inputs, labels et l'affichages des messages d'erreurs et des inputs invalides, vous pouvez utliser des themes, en configurant le fichier dans config / packages/ twig.yaml et en rajoutant le thème selectionné

```
twig:
    file_name_pattern: '*.twig'
    form_themes: ['bootstrap_5_horizontal_layout.html.twig']
```

Liste des form_themes ici

Exercice d’application :

1️⃣rajouter dans le dasboard les formulaires de creation / edition d’un produit. rendez le tout fonctionnel bien sur pour une creation/ modification en base tout d’abord dans un cas idéal (pas d’erreurs, c’est Lessieur)
2️⃣ Vous viendrez greffer des contraintes de validations simples sur certains champs et afficherez les messages en cas d’erreurs de saisie
3️⃣ A l’aide de la documentation essayez de faire un upload d’image

(spoil : allez devoir toucher a un fichier de configuration.yaml pour indiquer le répertoire de destination de l’upload)

Let’s go les dev ! 💪