

Les Formulaire

Lotfi KHEDIRI



Introduction



- ❑ La création et le traitement de formulaires HTML sont **difficiles** et **répétitifs**
 - ❑ -> Gérer le rendu des champs de formulaire HTML,
 - ❑ -> Assurer la validation des données soumises,
 - ❑ -> réaliser le mappage des données de formulaire en objets
 - ❑ -> et bien plus encore
- ❑ Symfony inclut un composant **Form** puissant qui fournit toutes ces fonctionnalités et bien d'autres pour des scénarios complexes.



Installation



Installer le composant Form

- ❑ `$composer require symfony/form`
- ❑ Le flux de travail recommandé lors de l'utilisation de formulaires Symfony est le suivant:
 - ❑ **Créez le formulaire**
 - ❑ **Rendez le formulaire** dans un template
 - ❑ **Traitez le formulaire** pour **valider** les données soumises, transformez-les en données PHP et faites-en quelque chose (par exemple, conservez-les dans une base de données).

Utilisation du composant Form



Créez le formulaire



Rendez le formulaire dans un template



Traitez le formulaire pour **valider** les données soumises, transformez-les en données PHP et faites-en quelque chose (par exemple, conservez-les dans une base de données).



Construction d'un formulaire

Deux méthodes pour créer un formulaire

- Il y a 2 façons de construire un formulaire
- 1. Créer le formulaire **dans le contrôleur** d'une page en utilisant un objet "**form builder**"
- 2. Créer le formulaire dans une **classe externe dédiée** qui dérive "**AbstractType**" (méthode recommandée)

Création d'un formulaire : Méthode

1

- Si le contrôleur dérive de AbstractController , on utilise la méthode helper **createFormBuilder()**

```
public function new(Request $request)
{
    // creates a produit object for this example
    $produit = new Product();
    $form = $this->createFormBuilder($produit)
        ->add('name', TextType::class )
        ->add('price', IntegerType::class )
        ->add('description', TextareaType::class )
        ->add('save', SubmitType::class , ['label' => 'Ajouter'])
        ->getForm();

    // ....
}
```

Les "Form types" prédéfinis de symfony



Création d'un formulaire : **Méthode**

2

- ❑ Symfony recommande de mettre le **moins de logique possible** dans les contrôleurs.
- ❑ Déplacer le code de création des formulaires complexes vers des classes dédiées : c'est les **classes de formulaires**.
- ❑ Ces classes sont **réutilisables** dans plusieurs actions ou services
- ❑ Les classes de formulaire implémentent l'interface **"FormTypeInterface"**. Cependant, il est préférable d'étendre la classe **"AbstractType"**, qui implémente déjà l'interface et fournit certains utilitaires.
- ❑ Pour créer un formulaire, on utilise la méthode helper **createForm()** dans le contrôleur.

Création d'un formulaire : **Méthode**

```
class ProductType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name')
            ->add('price')
            ->add('description')
            ->add('image')
            ->add('category')
            ->add('fournisseurs')
        ;
    }
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(['data_class' => Product::class,]);
    }
}
```

- La commande **bin/console make:form** permet de créer cette classe pour vous.



Rendu des Formulaires

Représentation visuelle du formulaire

- ▮ Au lieu de passer l'intégralité de l'objet de formulaire au modèle, utilisez la méthode **createView()** pour créer un autre objet avec la **représentation visuelle** du formulaire
- ▮ Passer cette représentation au template.

```
public function new(Request $request)
{
    $task = new Task();
    // ...

    $form = $this->createForm(TaskType::class, $task);

    return $this->render('task/new.html.twig', [
        'form' => $form->createView(),
    ]);
}
```

Affichage du formulaire dans le template

- ▢ Ensuite, utilisez des fonctions "helper" de formulaire pour rendre le contenu du formulaire.
- ▢ La fonction **form ()** rend tous les champs *et* les balises de début et de fin `<form>`
- ▢ Par défaut, la méthode du formulaire est POST
- ▢ L'URL cible (attribut action) est la même que celle affichée pour le formulaire.
- ▢ Les deux (action et method) peuvent être modifié ([lien](#))
- ▢ ce rendu n'est pas très flexible donc on aura besoin de plus de contrôle sur l'apparence de l'ensemble du formulaire

```
{# templates/task/new.html.twig #}  
{{ form(form) }}
```


Affichage du formulaire dans le template

▢ Rendu détaillé

```
{{ form_start(form) }}  
  <div class="my-custom-class-for-errors">  
    {{ form_errors(form) }}  
  </div>  
  
  <div class="row">  
    <div class="col">  
      {{ form_row(form.task) }}  
    </div>  
    <div class="col" id="some-custom-id">  
      {{ form_row(form.dueDate) }}  
    </div>  
  </div>  
{{ form_end(form) }}
```

Les thèmes de formulaire intégrés de Symfony


- ▮ Symfony possède des thèmes de formulaire intégrés qui incluent Bootstrap 3 et 4 et Foundation 5.
- ▮ Ces Thèmes permettent de styler rapidement tous les formulaires d'une application symfony.

```
# config/packages/twig.yaml  
  
twig:  
  
    form_themes: ['bootstrap_4_layout.html.twig']
```


- ▮ Possibilité de [créer votre propre thème de formulaire Symfony](#) .
- ▮ Symfony permet de *personnaliser la façon dont les champs sont rendus* avec plusieurs fonctions pour rendre chaque partie de champ séparément (widgets, étiquettes, erreurs, messages d'aide, etc.)



Traitement du formulaire



Modèle de commun de traitement de formulaire



```
public function new(Request $request)
{
    // just setup a fresh $task object (remove the example data)
    $task = new Task();

    $form = $this->createForm(TaskType::class, $task);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        // $form->getData() holds the submitted values
        // but, the original '$task' variable has also been updated
        $task = $form->getData();

        // ... perform some action, such as saving the task to the database
        // for example, if Task is a Doctrine entity, save it!
        // $entityManager = $this->getDoctrine()->getManager();
        // $entityManager->persist($task);
        // $entityManager->flush();

        return $this->redirectToRoute('task_success');
    }
}
```

Modèle commun de traitement de formulaire

Ce contrôleur suit un modèle commun pour la gestion des formulaires et a **trois** chemins possibles:

1. Lors du chargement initial, le formulaire n'a pas encore été soumis et **\$form->isSubmitted()** retourne **false**. Ainsi, le formulaire est créé et rendu;
2. Lorsque l'utilisateur soumet le formulaire, le **handleRequest()** reconnaît et réécrit immédiatement les données soumises dans les attributs de l'objet \$task. Cet objet est ensuite **validé**. S'il n'est pas valide, **isValid()** retourne **false** et le formulaire est rendu à nouveau, mais maintenant avec **des erreurs de validation**;
3. Lorsque l'utilisateur soumet le formulaire avec des données valides, les données soumises sont à nouveau écrites dans le formulaire, mais cette fois **isValid()** renvoie true. On a maintenant la possibilité d'effectuer certaines actions en utilisant l'objet \$task avant de rediriger l'utilisateur vers une autre page. (par exemple une page de "succès");



Validation des formulaires

Validation des formulaires

```
class Task
```

```
{
```

```
/**
```

```
 * @Assert\NotBlank
```

```
 */
```

```
public $task;
```

```
/**
```

```
 * @Assert\NotBlank
```

```
 * @Assert\Type("\DateTime")
```

```
 */
```

```
protected $dueDate;
```

```
}
```

- La validation se fait en ajoutant un ensemble de **règles** à la classe entité
- Chaque règle est appelée **contrainte**.
- Exemple, ajoutez des contraintes de validation afin que :
 - le champ **task** ne puisse **pas être vide** et
 - le champ **dueDate** ne puisse **pas être vide** et doit être un objet **DateTime** valide.
- [Documentation sur la validation](#)



Autres fonctionnalités de formulaire courantes

Options de type de formulaire

- ▮ Chaque type de formulaire ([les form types](#)) possède un certain nombre d'options pour le configurer ([référence aux types de formulaires Symfony](#)).
- ▮ Les deux options couramment utilisées sont **required** et **label**.
- ▮ L'option **required**, peut être appliquée à n'importe quel champ. Par défaut, cette option est définie sur **true**. Définir cette option à false pour désactiver la validation html5 côté client.
- ▮ L'option **required** n'effectue aucune validation côté serveur

```
->add('dueDate', \DateTime::class, [  
    'required' => false,  
)
```

- ▮ Par défaut, le label des champs de formulaire est la version *humanisée* du nom de l'attribut (user-> **User**; postalAddress-> **Postal Address**). Définissez l'option **label** sur les champs pour définir explicitement leurs labels:

```
->add('dueDate', \DateTime::class, [  
    // set it to FALSE to not display the label for this field  
    'label' => 'To Be Completed Before',  
)
```

Changer l'action et la méthode HTTP

- ▢ Modifier la méthode et/ou l'action d'un formulaire dans le **contrôleur**

```
public function new()  
{  
    // ...  
  
    $form = $this->createForm(TaskType::class, $task, [  
        'action' => $this->generateUrl('target_route'),  
        'method' => 'GET',  
    ]);  
  
    // ...  
}
```

- ▢ Modifier la méthode et/ou l'action d'un formulaire dans le **template**

```
{# templates/task/new.html.twig #}  
{{ form_start(form, {'action': path('target_route'), 'method': 'GET'}) }}
```

Désactiver la validation coté client

- ▮ La validation côté client peut être **désactivée** en ajoutant l'attribut **novalidate** à la balise **<form>**
- ▮ Cela est particulièrement utile lorsque vous souhaitez tester vos contraintes de validation côté serveur.

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


Champs non mappés

- ❑ Lors de la modification d'une entité via un formulaire, tous les champs du formulaire sont considérés comme des attribut de l'objet entité.
- ❑ Si on a besoin de champs supplémentaires dans le formulaire qui ne seront pas stockés dans l'objet (n'est pas un attribut de l'entité) alors définissez l'option **mapped** à **false** dans ces champs.

- ❑ Ces "champs non mappés" peuvent être définis et accessibles dans un contrôleur avec:

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('task')
        ->add('dueDate')
        ->add('agreeTerms', CheckboxType::class, ['mapped' => false])
        ->add('save', SubmitType::class)
    ;
}
```

```
$form->get('agreeTerms')->getData();
$form->get('agreeTerms')->setData(true);
```



To Learn

- Comment personnaliser le rendu des formulaires
- Les Form types de symfony (types prédéfinis des champs)
- Les contraintes de validation prédéfinies de symfony
- How to Upload Files
- How to Implement CSRF Protection