



Composant de création des formulaire dans Symfony

UP Web

AU: 2019/2020

Plan



- Introduction
- Création du Formulaire
- Fonctions de rendu de formulaire: TWIG
- Soumission du formulaire: Controller
- Exemples de types de champs

Introduction



- Symfony intègre un composant Form qui aide à gérer les formulaires .
- Symfony offre une grande variété de façons de personnaliser le rendu d'un formulaire.
- Le composant Form permet de créer des formulaires:
 - Directement dans le contrôleur ➔ Convient parfaitement si vous n'avez pas besoin de réutiliser le formulaire ailleurs.
 - Dans une classe à part.


Recommandation: Pour l'organisation et la réutilisation, définir chaque formulaire dans sa propre classe.

Creation du formulaire



■ Génération du formulaire dans le contrôleur:

```
public function AjoutAction(Request $request)
{
    $Formation = new Formation();
    $form = $this->createFormBuilder($Formation)
        ->add( child: 'titre', type: TextType::class)
        ->add( child: 'description', type: TextareaType::class)
        ->add( child: 'dateDebut', type: DateType::class)
        ->add( child: 'dateFin', type: DateType::class)
        ->add( child: 'nbrParticipant', type: IntegerType::class)
        ->add( child: 'id_Club', type: EntityType::class, array(
            'class'=>'GestionClubBundle:Club',
            'choice_label'=>'nom',
            'multiple'=>false
        ))
        ->add( child: 'Ajouter', type: SubmitType::class,
            ['label'=>'Ajouter Formation'])
    ->getForm();
    return $this->render( view: '@GestionClub/Club/ajout_club.html.twig', [
        'f' => $form->createView(),
    ]);
}
```



Générer le rendu du formulaire

Creation du formulaire



■ Génération de la classe formulaire PHP:

```
doctrine:generate:form ClubBundle:Formation
```

```
?php
```

```
namespace GestionClubBundle\Form;
```

```
use ...
```

```
class ClubType extends AbstractType
```

```
public function buildForm(FormBuilderInterface $builder,  
    array $options)
```

```
{
```

```
    $builder
```

```
        ->add( child: 'nom')
```

```
        ->add( child: 'description')
```

```
        ->add( child: 'adresse')
```

```
        ->add( child: 'domaine');
```

```
    /**
```

```
     * {@inheritdoc}
```

```
     */
```

```
public function configureOptions(OptionsResolver $resolver)
```

```
{
```

```
    $resolver->setDefaults(array(  
        'data_class' => 'GestionClubBundle\Entity\Club'
```

```
    ));
```

```
    }
```

```
<?php
```

```
namespace GestionClubBundle\Controller;
```

```
use GestionClubBundle\Entity\Club;
```

```
use GestionClubBundle\Form\ClubType;
```

```
class ClubController extends Controller
```

```
{
```

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

```
    {
```

```
        $club = new Club();
```

```
        $form = $this->createForm(ClubType::class, $club);
```

```
    }
```

```
}
```

➔ Réutilisation du formulaire

Creation du formulaire



- Le composant de création des formulaires de Symfony permet d'ajouter des boutons sous forme de champs dans le formulaire, ce qui permet de simplifier le modèle de rendu du formulaire. Mais si on ajoute les boutons directement dans la classe de FormType, cela limiterait effectivement la portée de celle ci.

```
<?php

namespace GestionClubBundle\Form;
use ...

class ClubType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder,
                             array $options)
    {
        $builder
            ->add( child: 'Ajouter', type: SubmitType::class,
                ['label'=>'Ajouter Club']);
    } /**
     * {@inheritdoc}
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'GestionClubBundle\Entity\Club'
        ));
    }
}
```

Creation du formulaire



- En cas de réutilisation du formulaire, le libellé du bouton serait faux → Configurer des boutons de formulaire dans le contrôleur

Rq: Le fait de déclarer le bouton dans le contrôleur, c'est également une erreur importante, car on mélange le balisage de présentation (étiquettes, classes CSS, etc.) avec du code PHP pur. La séparation des préoccupations est toujours une bonne pratique à suivre.

→ Mettez donc tous les éléments relatifs à la vue dans la couche de vue,

```
<?php

namespace GestionClubBundle\Controller;

use ...

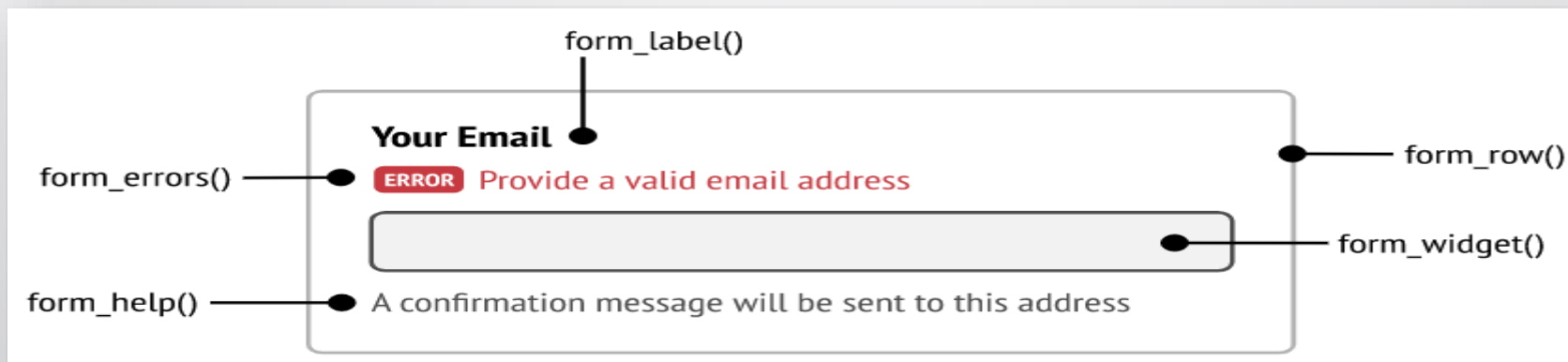
class ClubController extends Controller
{
    public function AjoutClubAction(Request $request)
    {
        $club=new Club();
        $Form=$this->createForm( type: ClubType::class,$club);
        $Form->add( child: 'Ajouter', type: SubmitType::class,
            ['label' => 'Ajouter Club',
             'attr'  => ['class' => 'Ajout'],
            ]);
    }
}
```

Fonctions de rendu de formulaire

TWIG



- Un seul appel à la fonction `{{ form(f) }}` Twig est suffisant pour afficher un formulaire complet, y compris tous ses champs et ses messages d'erreur.
NB: « **f** » est une variable transmise par le contrôleur et créée en appelant la méthode `$f->createView()`
- On peut restituer les différentes parties du formulaire afin de les personnaliser en ajoutant des éléments et des attributs HTML en utilisant les fonctions twig : `form_start()`, `form_end()`, `form_errors()` et `form_row()`



Fonctions de rendu de formulaire

TWIG



- `{{ form_start() }}` → Rend la balise de début du formulaire
- `{{ form_widget() }}` → Génère le rendu de tous les champs, y compris l'élément de champ lui-même, une étiquette et les éventuels messages d'erreur de validation du champ.
- `{{ form_end() }}` → Rend la balise de fin du formulaire et tous les champs qui n'ont pas encore été rendus, au cas où on aura rendu nous-même chaque champ. Ceci est utile pour afficher les champs cachés et tirer parti de la protection automatique **CSRF**.

```
{{ form_start(f) }}  
{{ form_widget(f) }}  
{{ form_end(f) }}
```

- **NB:** Cross-site request forgery (CSRF) est une méthode par laquelle un utilisateur malveillant tente de faire en sorte que vos utilisateurs légitimes soumettent sans le savoir des données qu'ils ne souhaitent pas soumettre. Heureusement, les attaques CSRF peuvent être prévenues en utilisant un jeton CSRF dans vos formulaires.

Variables de rendu de formulaire TWIG



- Certaines des fonctions Twig permettent de passer des variables pour configurer leur comportement.

```
{{ form_label(f.id_Club, 'Nom_Club') }}
```

Titre:	<input type="text"/>
Description	<input type="text"/>
Date debut	2014 ▼ Jan ▼ 1 ▼
Date fin	2014 ▼ Jan ▼ 1 ▼
Nbr participant	<input type="text"/>
Id club	Web ▼
<input type="button" value="Ajouter"/>	



Titre:	<input type="text"/>
Description	<input type="text"/>
Date debut	2014 ▼ Jan ▼ 1 ▼
Date fin	2014 ▼ Jan ▼ 1 ▼
Nbr participant	<input type="text"/>
Nom_Club	Web ▼
<input type="button" value="Ajouter"/>	

```
{{ form_label(f.description, 'La description:',  
  {'label_attr':{'class':'desc'}}) }}
```

```
{{ form_label(f.description, null, {  
  'label': 'La description est:',  
  'label_attr':{'class':'desc'}}) }}
```

Variables de rendu de formulaire

TWIG



- Ajouter le fait qu'on peut avoir action= dans form

```
{{ form_widget(f.description, {'attr': {'class': 'desc'}}) }}
```

Attribuer une classe à l'input de description

```
{{ form(f, {'method': 'GET'}) }}
```

```
{{ form_start(f, {'method': 'GET'}) }}
```

Modifier la méthode de récupération
des données du formulaire « f »

- Par défaut, le formulaire soumettra une demande POST au même contrôleur que celui-ci.

Soumission formulaire

Contrôler



```
public function AjoutFormationAction(Request $request)
{
    $formation=new Formation();
    $Form=$this->createForm( type: FormationType::class,$formation);
    $Form->handleRequest($request);
    if($Form->isSubmitted() && $Form->isValid()){
        $em=$this->getDoctrine()->getManager();
        $em->persist($formation);
        $em->flush();
        return $this->redirect($this->generateUrl(
            route: '_afficher_formation',
            ['id' => $formation->getId()]
        ));
    }
    return $this->render( view: '@GestionClub/Formation/ajout_formation.html.twig',
        array('f'=>$Form->createView()));
}
```

Création du Formulaire: Placer la logique de formulaire dans sa propre classe signifie que le formulaire peut être réutilisé ailleurs dans votre projet. → C'est la meilleure façon de créer des formulaires

Recommandation :

- Utiliser une seule action pour le rendu du formulaire ainsi que le traitement de l'envoi du formulaire.
- Il est nécessaire d'appeler `$form->isSubmitted()` dans l'instruction `if` avant d'appeler `isValid()`.
- L'appel de `isValid()` avec un formulaire non soumis est obsolète depuis la version 3.2 et lève une exception
- dans la version 4.0.

Soumission formulaire

Contrôler



```
public function AjoutFormationAction(Request $request)
{
    $formation=new Formation();
    $Form=$this->createForm( type: FormationType::class,$formation);
    $Form->handleRequest($request);
    if($Form->isSubmitted() && $Form->isValid()){
        $em=$this->getDoctrine()->getManager();
        $em->persist($formation);
        $em->flush();
        return $this->redirect($this->generateUrl(
            route: '_afficher_formation',
            ['id' => $formation->getId()]
        ));
    }
    return $this->render( view: '@GestionClub/Formation/ajout_formation.html.twig',
        array('f'=>$Form->createView()));
}
```

Traiter la soumission
du formulaire et
reconnaître que le
formulaire n'a pas
été soumis et ne fait
rien

Recommandation :

- Utiliser une seule action pour le rendu du formulaire ainsi que le traitement de l'envoi du formulaire.
- Il est nécessaire d'appeler `$form->isSubmitted()` dans l'instruction `if` avant d'appeler `isValid()`.
- L'appel de `isValid()` avec un formulaire non soumis est obsolète depuis la version 3.2 et lève une exception
- dans la version 4.0.

Soumission formulaire

Contrôler



```
public function AjoutFormationAction(Request $request)
{
    $formation=new Formation();
    $Form=$this->createForm( type: FormationType::class,$formation);
    $Form->handleRequest($request);
    if ($Form->isSubmitted() && $Form->isValid()) {
        $em=$this->getDoctrine()->getManager();
        $em->persist($formation);
        $em->flush();
        return $this->redirect($this->generateUrl(
            route: '_afficher_formation',
            ['id' => $formation->getId()]
        ));
    }
    return $this->render( view: '@GestionClub/Formation/ajout_formation.html.twig',
        array('f'=>$Form->createView()));
}
```

envoi false si le
formulaire n'a pas
été soumis.

Renvoie false si le formulaire n'est pas
valide et le formulaire est à nouveau
rendu, mais avec des erreurs de
validation

Renvoie l'objet
view du formulaire

Recommandation :

- Utiliser une seule action pour le rendu du formulaire ainsi que le traitement de l'envoi du formulaire.
- Il est nécessaire d'appeler `$form->isSubmitted()` dans l'instruction `if` avant d'appeler `isValid()`.
- L'appel de `isValid()` avec un formulaire non soumis est obsolète depuis la version 3.2 et lève une exception dans la version 4.0.

Exemples de types de champs dans un formulaire SMF



Champs de texte	TextType
	TextareaType
	EmailType
	IntegerType
	MoneyType
	NumberType
	PasswordType
	PercentType
	SearchType
	UrlType
	RangeType
	TelType
	ColorType
	HiddenType

Champs de choix	ChoiceType
	EntityType
	CountryType
	LanguageType
	LocaleType
	TimezoneType
	CurrencyType
	CheckboxType
	FileType
	RadioType

Champs de date et temps	DateType
	DateIntervalType
	DateTimeType
	TimeType
	BirthdayType



- **Atelier:**

Référence



- ❑ https://symfony.com/doc/current/form/form_customization.html
- ❑ <https://symfony.com/doc/3.4/forms.html>