المعهد العالي للدراسات التكنولوجية بمدنين
Institut Supérieur des Etudes Technologiques de Médenine

Atelier Framework Côté Serveur

TP5

Les formulaires

(Symfony 6)

Enseignante : K.MECHLOUCH	Classe : L2DSI

Pour travailler avec des formulaires Symfony il faut :

a. construire le formulaire dans un contrôleur Symfony ou en utilisant une classe de formulaire dédiée ;
b. afficher le formulaire dans un modèle afin que l'utilisateur puisse le modifier et le soumettre ;
c. traiter le formulaire pour valider les données soumises, transformer-le en données PHP et faire quelque chose avec (par exemple, persistez-le, dans une base de données).

1. Installation

Exécutez cette commande pour installer la fonctionnalité de formulaire avant de l'utiliser :

```
composer require symfony/form
```

2. Création

Symfony fournit un objet ***form builder*** qui permet de décrire les champs du formulaire à l'aide d'une interface. Plus tard, ce générateur crée l'objet de formulaire réel utilisé pour rendre et traiter le contenu.
Pour créer un formulaire dans l'action d'un contrôleur il suffit d'invoquer la méthode ***createFormBuilder()*** et d'ajouter les champs désirés avec la méthode ***add()*** :

```
$form=$this->createFormbuilder()
        ->add('nomduchamp', Type de champ, [ options ])
```

Les types de champ disponibles sont définis dans la documentation de Symfony : https://symfony.com/doc/current/reference/forms/types.html

On veut maintenant créer un formulaire de contact.

Créez l'entité suivante :

```
// src/Entity/Contact.php
namespace App\Entity;

class Contact
{
    protected $nom;
    protected $email;
    protected $message;

    public function getNom(): string
    {
        return $this->nom;
    }
    public function setNom(string $nom): void
    {
        $this->nom = $nom;
    }
    public function getEmail(): string
    {
        return $this->email;
    }
    public function setEmail(string $email): void
    {
        $this->email = $email;
    }
    public function getMessage(): string
    {
        return $this->message;
    }
    public function setMessage(string $message): void
    {
        $this->message = $message;
    }
}
```

2.1.Création dans un contrôleur

Créer le contrôleur suivant :

```
// src/Controller/ContactController.php
namespace App\Controller;

use App\Entity\Contact;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;

class ContactController extends AbstractController
{
```

```
……..
public function addContact()
   {       // créer un objet contact
      $contact = new Contact();
      $form = $this->createFormBuilder($contact)
         ->add('nom', TextType::class)
         ->add('email', TextType::class)
         ->add('message', TextareaType::class)
         ->add('envoyer', SubmitType::class, ['label' => 'Envoyer'])
         ->getForm();
      // ...
   }
}
```

2.2.Création dans une classe indépendante

Les formulaires indépendants vont pouvoir être réutilisés dans plusieurs actions sans qu'il y ait besoin de les redéfinir. On pourra également imbriquer ces formulaires, faire de l'héritage de formulaires, etc.
Les classes de formulaire sont des ***form type*** qui implémentent ***FormTypeInterface***. Cependant, il est préférable d'étendre ***AbstractType***, qui implémente déjà l'interface et fournit quelques utilitaires.

Pour créer un formulaire, vous pouvez servir d'une commande sur le terminal :

```
php bin/console make:form
```

ou créer la classe manuellement :

```
// src/Form/Type/ContactType.php
namespace App\Form\Type;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\FormBuilderInterface;

class ContactType extends AbstractType
{
   public function buildForm(FormBuilderInterface $builder, array $options): void
   {
      $builder
         ->add('nom', TextType::class)
         ->add('email', TextType::class)
         ->add('message', TextareaType::class)
         ->add('envoyer', SubmitType::class)
      ;
   }
}
```

```
class ContactController extends AbstractController
{
   public function new()
   {       // créer un objet contact
       $contact = new Contact();
       $form = $this->createForm(ContactType::class,$contact);
      // ...
   }
}
```

3. Affichage

Maintenant que le formulaire a été créé, la prochaine étape consiste à faire l'affichage :

```
class ContactController extends AbstractController
{
   public function new(Request $request)
   {       // créer un objet contact
       $contact = new Contact();
       $form = $this->createForm(ContactType::class,$contact);
       return $this->render('contact/new.html.twig', [
          'form' => $form,
       ]);

   }
}
```

L'avantage avec Symfony, c'est qu'on peut développer un formulaire en PHP sans se soucier de la partie HTML. Nous nous sommes affranchis de l'écriture des balises <*input*>, qui sont lourdes et fastidieuses.
L'instruction :

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

s'occupe de tout.
Cette instruction utilise un thème par défaut pour afficher le formulaire.
Il est possible de modifier ce thème par défaut et, par exemple, d'utiliser un thème intégrant Bootstrap.
Pour ce faire, il faut modifier le fichier ***config/packages/twig.yaml*** et y préciser le thème de Bootstrap 5 :

```
# config/packages/twig.yaml
twig:
   form_themes: ['bootstrap_5_layout.html.twig']
```

4. Traitement des données du formulaire

Le formulaire étant au point, il s'agit maintenant de nous préoccuper de la récupération des informations saisies par l'utilisateur et de l'insertion en base de données.
En Symfony, tout se passe dans la même action, la création et le traitement du formulaire.

Pour récupérer les données dans l'entité associée, il faut utiliser la méthode ***handleRequest()*** en lui passant l'objet ***$request*** :

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

Puis il faut tester l'existence de la méthode ***Post*** dans l'objet ***$request***. Il est conseillé de tester également la méthode ***isValid()***. Cette méthode contrôle que toutes les données du formulaire vérifient les contraintes de validation.

```
if($request->isMethod('post') && $form->isValid() ){...}
```

À l'intérieur de l'instruction de contrôle *if*, insérer les données dans la base de données :

```
 $entityManager->persist(….);
$entityManager->flush();
```

```
// src/Controller/ContactController.php

// ...
use Symfony\Component\HttpFoundation\Request;

class ContactController extends AbstractController
{……
   public function new(Request $request): Response
   {
      // créer un objet contact
      $contact = new Contact();

      $form = $this->createForm(ContactType::class, $contact);

      $form->handleRequest($request);
      if ($form->isSubmitted() && $form->isValid()) {
         // récupérer les données envoyées
         $contact= $form->getData();

         // traitement: par exemple insertion dans la base de données

         return new Response('success');
      }

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

- **Validation**

La validation est effectuée en ajoutant un ensemble de règles, appelées contraintes (de validation), à une classe. Vous pouvez les ajouter soit à la classe d'entité, soit à la classe de formulaire.
Avant d'utiliser la validation, exécuter :

```
composer require symfony/validator
```

Ajouter des contraintes à l'entité *Contact*, de sorte que le champ email soit valide et le champ message ne puissent pas être vides.

```
use Symfony\Component\Validator\Constraints as Assert;
class Contact
{
   protected $nom;
  /**
    * @Assert\Email
    */
    protected $email;
   /**
    * @Assert\NotBlank
    */
    protected $message;
…..
  }
```

On peut ajouter certaines options pour les champs, lors de la création du formulaire :

```
->add('message', TextareaType::class, [
    'required' => false,
])
->add('email', TextType::class, [

    'label' => 'votre email',
])
->add('agreeTerms', CheckboxType::class, ['mapped' => false])
```

- **Modifier la route et la méthode HTTP**

Par défaut, le formulaire sera soumis via une requête HTTP POST à la même URL sous laquelle le formulaire a été consulté.
Pour modifier la route et la méthode :

- utiliser ***setAction()*** et ***setMethod()*** :

```
$form = $this->createFormBuilder($contact)
                    ->setAction($this->generateUrl('target_route'))
                    ->setMethod('GET')
        // ...
                    ->getForm();
```

- Lors de la construction du formulaire dans une classe, transmettez l'action et la méthode en tant qu'options de formulaire :

```
$form = $this->createForm(TaskType::class, $task, [
        'action' => $this->generateUrl('target_route'),
        'method' => 'GET',
    ]);
```

- Remplacez l'action et la méthode en les transmettant aux fonctions ***form()*** ou ***form_start()*** :

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