Formulaire symfony par la pratique

1. ## Objectifs

2. ## Présentation

   - Commençons par explorer cette section fascinante sur la création de formulaires ! Symfony offre un système robuste pour la gestion des données soumises par les formulaires.
   - Symfony propose une commande pour générer automatiquement un formulaire. Ce formulaire n'est pas généré en HTML, mais en une classe PHP. À partir de cette classe, nous pourrons apporter toutes les modifications nécessaires telles que les champs obligatoires, les classes CSS, et les validations des données.
   - Lorsque nous générons un formulaire, nous avons deux options : **soit nous le baser sur une entité pour enregistrer** ou mettre à jour des données dans la table associée, **soit le générer sans lien avec une entité spécifique** pour créer notre formulaire librement.
   - Dans un premier temps, voyons comment créer un formulaire lié à une entité.
   - Supposons que nous ayons une entité nommée "**Etudiant.php**" qui ressemble à ceci :

```php
<?php

namespace App\Entity;

use App\Repository\EtudiantRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: EtudiantRepository::class)]
class Etudiant
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 100, nullable: true)]
    private ?string $prenom = null;
```

apcpedagogie.com

```
    #[ORM\Column(length: 100, nullable: true)]
    private ?string $nom = null;

    #[ORM\Column(length: 255, nullable: true)]
    private ?string $adresse = null;

    #[ORM\Column(length: 50, nullable: true)]
    private ?string $gouvernorat = null;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getPrenom(): ?string
    {
        return $this->prenom;
    }

    public function setPrenom(?string $prenom): static
    {
        $this->prenom = $prenom;

        return $this;
    }

    public function getNom(): ?string
    {
        return $this->nom;
    }

    public function setNom(?string $nom): static
    {
        $this->nom = $nom;

        return $this;
    }

    public function getAdresse(): ?string
    {
        return $this->adresse;
    }
```

apcpedagogie.com

```
    public function setAdresse(?string $adresse): static
    {
        $this->adresse = $adresse;

        return $this;
    }

    public function getGouvernorat(): ?string
    {
        return $this->gouvernorat;
    }

    public function setGouvernorat(?string $gouvernorat): static
    {
        $this->gouvernorat = $gouvernorat;

        return $this;
    }
}
```

3. Créer un formulaire Symfony

- Générons un formulaire à partir cette entité, en utilisant le terminal de commande : **`symfony console make:form EtudiantType`** Vous pouvez aussi utiliser une autre commande **`php bin/console make:form EtudiantType`**
- Je donne le nom de "**EtudiantType**" en **UpperCamelCase** à mon formulaire. Sachez que le nom est libre et par convention, se termine par "**Type**". J'ai choisi le nom "**EtudiantType**" de façon à comprendre rapidement que ce formulaire est relié à l'entité "**Etudiant.php**".
- Ensuite, une question vous sera posée, vous demandant si vous désirez utiliser une entité ou non. Voulant utiliser "**Etudiant.php**", je réponds "**Etudiant**". Il n'est pas besoin de spécifier l'extension, "**Etudiant**" étant le nom de la classe PHP contenue dans mon entité.
- En retour, on me confirme le succès de la création du formulaire et on me donne l'endroit où se situe le fichier nouvellement créé : dans "**src/Form/**", sous le nom "**EtudiantType.php**". En regardant le contenu de ce fichier, nous pouvons voir qu'il contient différents `add()` intégrant les noms des propriétés de l'entité choisie.

```
<?php
```

apcpedagogie.com

```
namespace App\Form;

use App\Entity\Etudiant;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class EtudiantType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('prenom')
            ->add('nom')
            ->add('adresse')
            ->add('gouvernorat')
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Etudiant::class,
        ]);
    }
}
```

- Chaque `add()` représente un champ du formulaire et est de type texte pour le moment. Bien évidemment, nous pouvons modifier les types selon nos besoins.

4. Types de champs

```
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
class EtudiantType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('prenom', TextType::class)
            ->add('nom', TextType::class)
```

apcpedagogie.com

```
            ->add('adresse', TextareaType::class)
            ->add('gouvernorat')
            ->add('save', SubmitType::class)
        ;
    }
```

- En second paramètre des `add()`, j'ai donc ajouté deux classes qui sont TextType et TextareaType. Leur nom suffit à comprendre de quoi il retourne. Pour "name", j'ai précisé que je veux le type texte, bien que comme je vous l'ai dit il s'agit du type par défaut ! J'ai fait cela tout simplement parce que pour accéder au troisième argument, je suis obligé de spécifier le second (fonction PHP basique).
- Pensez bien à ajouter les namespaces nécessaires en haut du document.

```
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
```

5. Options

- Le troisième argument de nos `add()` est un tableau associatif. Il existe plusieurs options disponibles, mais nous allons nous concentrer sur les plus utilisées.
- Il existe plusieurs options disponibles pour personnaliser les champs de formulaire dans Symfony :
  - **required** : Permet de spécifier si un champ est obligatoire ou non. Par défaut, la valeur est true.
  - **label** : Permet de définir le libellé du champ, qui sera affiché à côté de celui-ci dans le formulaire.
  - **label_attr** : Permet de spécifier des attributs HTML supplémentaires pour le libellé du champ.
  - **attr**: Permet de spécifier des attributs HTML supplémentaires pour le champ lui-même.
  - **help**: Permet d'ajouter un message d'aide ou de description pour le champ.
  - **empty_data** : Permet de définir une valeur par défaut pour le champ s'il est vide.
  - **constraints**: Permet de spécifier des contraintes de validation pour le champ.
  - Ces options peuvent être utilisées pour personnaliser le comportement et l'apparence des champs de formulaire selon les besoins de votre application.

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

apcpedagogie.com

```
        $builder
            ->add('prenom', TextType::class, [
                'required' => true,
                'label' => "Prénom de l'étudiant"
             ])
            ->add('nom', TextType::class, [
                'required' => true,
                'label' => "Nom de l'étudiant"
             ])
            ->add('adresse', TextareaType::class, [
                'required' => true,
                'label' => "Adresse de l'étudiant"
             ])
            ->add('gouvernorat')
            ->add('save', SubmitType::class)
        ;
    }
```

6. Contraintes de validations

- Les contraintes de validation sont essentielles pour garantir l'intégrité et la validité des données côté serveur. Voici quelques-unes des contraintes de validation les plus couramment utilisées dans Symfony :

  NotBlank: Le champ ne doit pas être vide. **Length**: Le champ doit avoir une longueur spécifique. **Email**: Le champ doit contenir une adresse e-mail valide. **Regex**: Le champ doit correspondre à un motif regex spécifique. **Range** : Le champ doit être dans une plage de valeurs spécifique. **Choice**: Le champ doit être une des valeurs spécifiées. **DateTime** : Le champ doit être une date et/ou une heure valide. **Type**: Le champ doit être d'un certain type (par exemple, integer, string, array, etc.).
- En utilisant ces contraintes, vous pouvez définir des règles de validation pour chaque champ de votre formulaire, assurant ainsi que seules les données valides sont acceptées et traitées côté serveur. Cela contribue à renforcer la sécurité et la fiabilité de votre application.

```
public function buildForm(FormBuilderInterface $builder, array $options): void
   {
       $builder
           ->add('prenom', TextType::class, [
               'required' => true,
```

```
                'label' => "Prénom de l'étudiant",
                'constraints' => [
                   new NotBlank([
                      'message' => 'Veuillez saisir votre prénom'
                   ]),
                   new Length([
                      'min' => 6,
                      'minMessage' => 'Le prénom doit contenir au minimum {{ limit
}} caractères'
                   ]),
                ]
             ])
            ->add('nom', TextType::class, [
                'required' => true,
                'label' => "Nom de l'étudiant",
                'constraints' => [
                   new NotBlank([
                      'message' => 'Veuillez saisir un nom'
                   ]),
                   new Length([
                      'min' => 6,
                      'minMessage' => 'Le nom doit contenir au minimum {{ limit }}
caractères'
                   ]),
                ]
             ])
            ->add('adresse', TextareaType::class, [
                'required' => true,
                'label' => "Adresse de l'étudiant",
                'constraints' => [
                   new NotBlank([
                      'message' => 'Veuillez saisir une adresse'
                   ])
                ]


             ])
            ->add('gouvernorat')
            ->add('save', SubmitType::class)
        ;
    }
```

apcpedagogie.com

7. Formulaire et relations de table

- Pour gérer une relation dans un formulaire Symfony, il est nécessaire de configurer le champ associé à cette relation de manière appropriée.
- Dans notre cas, puisque le champ "gouvernorat" dans notre entité est une relation vers une autre table "gouvernorats", nous devons configurer le champ dans notre formulaire pour afficher les gouvernorats disponibles dans un menu déroulant.
- Voici comment nous pouvons le faire dans notre méthode buildForm :

```
->add('gouvernorat', EntityType::class, [
                'required' => true,
                'label' => 'Choisir le gouvernorat',
                'class' => Gouvernorats::class,
                'choice_label' => 'libelle_gouv',
                'constraints' => [
                  new NotBlank([
                    'message' => 'Veuillez choisir une ville'
                  ])
                ]
              ])
```

- Je vais utiliser la classe **EntityType**. Je dois fournir deux options avec cette classe :
  - **Premièrement**, class, où je précise quelle entité je souhaite utiliser. Ma propriété "gouvernorat" étant une relation sur l'entité "Gouvernorats", je lui donne cette dernière.
  - **Deuxièmement**, l'option `choices_label`, qui permet de savoir quelles informations afficher dans le menu déroulant. Il faut ici choisir le nom de la propriété qui vous semble la plus appropriée. Pour ma part, dans mon entité "Gouvernorats.php", c'est la propriété "libelleGouv" qui contient le nom des différentes gouvernorats, je vais donc la sélectionner.
  - Comme pour les autres champs, j'ai ajouté l'obligation de choisir une ville avec `required`, j'ai entré un nouveau label et une contrainte de validation, pour éviter qu'un petit malin me retourne une valeur vide.
  - Avec ce code, vous obtiendrez un menu déroulant contenant les noms de toutes les gouvernorats présentes en table "gouvernorats" et vous aurez en valeur leur ID. Au moment de l'insertion, Symfony sera capable d'insérer le bon ID dans la colonne "gouvernorat_id". ?
  - Voilà mon code au complet :

apcpedagogie.com

```
<?php

namespace App\Form;
use App\Entity\Gouvernorats;
use App\Entity\Etudiant;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Validator\Constraints\NotBlank;
use Symfony\Component\Validator\Constraints\Length;

class EtudiantType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options):
void
    {
        $builder
            ->add('prenom', TextType::class, [
                'required' => true,
                'label' => "Prénom de l'étudiant",
                'constraints' => [
                   new NotBlank([
                      'message' => 'Veuillez saisir votre prénom'
                   ]),
                   new Length([
                      'min' => 6,
                      'minMessage' => 'Le prénom doit contenir au minimum {{
limit }} caractères'
                   ]),
                ]
             ])
            ->add('nom', TextType::class, [
                'required' => true,
                'label' => "Nom de l'étudiant",
                'constraints' => [
                   new NotBlank([
                      'message' => 'Veuillez saisir un nom'
                   ]),
```

```
                    new Length([
                        'min' => 6,
                        'minMessage' => 'Le nom doit contenir au minimum {{
limit }} caractères'
                    ]),
                ]
             ])
            ->add('adresse', TextareaType::class, [
                'required' => true,
                'label' => "Adresse de l'étudiant",
                'constraints' => [
                    new NotBlank([
                        'message' => 'Veuillez saisir une adresse'
                    ])
                ]

             ])
            ->add('gouvernorat', EntityType::class, [
                'required' => true,
                'label' => 'Choisir le gouvernorat',
                'class' => Gouvernorats::class,
                'choice_label' => 'libelle_gouv',
                'constraints' => [
                  new NotBlank([
                    'message' => 'Veuillez choisir une ville'
                  ])
                ]
              ])
            ->add('save', SubmitType::class)
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Etudiant::class,
        ]);
    }
}
```

apcpedagogie.com

8. Afficher le formulaire

- Pour afficher votre formulaire dans une vue Twig, vous devez d'abord **créer un contrôleur** avec une méthode qui génère et **affiche le formulaire**, puis créer une vue Twig pour afficher le formulaire.
- Voici comment vous pouvez procéder :

1. **Créez une méthode dans votre contrôleur pour générer et afficher le formulaire :**

   - Créer votre controleur avec la commande :**`php bin/console make:controller EtudiantController`**

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class EtudiantController extends AbstractController
{
    #[Route('/etudiant', name: 'app_etudiant')]
    public function index(): Response
    {
        return $this->render('etudiant/index.html.twig', [
            'controller_name' => 'EtudiantController',
        ]);
    }
}
```

   - Maintenant votre controller est crée vous pouvez le modifier comme suit:

```
<?php
namespace App\Controller;

use App\Entity\Etudiant;
use App\Form\EtudiantType;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
```

apcpedagogie.com

```
use Symfony\Component\Routing\Attribute\Route;

class EtudiantController extends AbstractController
{
    #[Route('/etudiant', name: 'app_etudiant')]
    public function index(): Response
    {
        return $this->render('etudiant/index.html.twig', [
            'controller_name' => 'EtudiantController',
        ]);
    }
    #[Route('/nouvel-etudiant', name: 'nouvel_etudiant')]
    public function nouvelEtudiant(Request $request)
    {
        $etudiant = new Etudiant();
        $form = $this->createForm(EtudiantType::class, $etudiant);
        $form->handleRequest($request);
        return $this->render(
            'etudiant/nouvel_etudiant.html.twig',
            [
                'formNouvelEtudiant' => $form->createView()
            ]
        );
    }
}
```

- Détaillons un peu le code pour mieux comprendre.
- **Instanciation de l'entité Etudiant :**
  - Dans un premier temps, vous devez instancier un nouvel objet de votre entité Etudiant. Cela se fait généralement en utilisant le mot-clé new suivi du nom de votre entité :**`$etudiant = new Etudiant();`**
  - Cette étape crée une nouvelle instance de votre **entité Etudiant**, qui sera utilisée pour stocker les données soumises par le formulaire.
- **Création du formulaire :**
  - Ensuite, vous devez créer un formulaire pour saisir les informations de l'étudiant. Vous pouvez utiliser la méthode `createForm()` de votre contrôleur pour cela. Vous devez spécifier le type de formulaire que vous avez créé (dans votre cas, `EtudiantType`) et passer l'objet **Etudiant** que vous avez créé précédemment :**`$form = $this->createForm(EtudiantType::class, $etudiant);`**
  - Cette ligne crée un formulaire basé sur le type de formulaire EtudiantType, en associant l'objet Etudiant que vous avez créé.

apcpedagogie.com

- **Gestion de la soumission du formulaire :**
  - Une fois que le formulaire est créé, vous devez gérer la soumission du formulaire. Cela se fait en appelant la méthode `handleRequest()` du formulaire et en passant l'objet de la requête en cours (`$request`) : **`$form->handleRequest($request);`**
  - Cette étape permet au formulaire de traiter les données soumises et de les associer à l'objet Etudiant.
- **Rendu du formulaire dans la vue Twig :**
  - Enfin, vous devez rendre le formulaire dans votre vue Twig associée. Pour cela, vous utilisez la méthode `createView()` du formulaire pour créer une représentation du formulaire qui peut être utilisée dans votre template Twig. Vous pouvez ensuite passer cette représentation du formulaire à votre vue Twig :

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

  - Cette ligne de code rend le formulaire dans la vue **nouvel_etudiant.html.twig**, en passant la représentation du formulaire sous forme de variable form.

> ***En résumé, ces étapes vous permettent de créer un formulaire, de le lier à une entité, de le gérer lors de sa soumission et de le rendre dans une vue Twig pour que l'utilisateur puisse le remplir et soumettre ses données. Assurez-vous d'adapter les noms des entités, des formulaires, des routes et des vues selon votre propre structure d'application.***

2. **Créez une vue Twig pour afficher le formulaire :**

```
{% extends 'base.html.twig' %}

{% block title %}Nouvel Etudiant!{% endblock %}

{% block body %}
```

```
<h1>Nouvel Etudiant</h1>


{% endblock %}
```

3. **Afficher le formulaire dans la vue**

   - Il reste maintenant à afficher le formulaire dans la vue. On peut le faire tout simplement avec le code suivant :

```
{% extends 'base.html.twig' %}

{% block title %}Nouvel Etudiant!{% endblock %}

{% block body %}

<h1>Nouvel Etudiant</h1>


{{ form(formNouvelEtudiant) }}
{% endblock %}
```

   - On va utiliser une fonction Twig appelée `form()` et lui passer en paramètre la variable "formNouvelEtudiant", envoyée depuis notre contrôleur, afin d'afficher visuellement le formulaire et de générer son code HTML.

4. **Vérifier si le formulaire a été soumis et si les données retournées sont correctes.**

   - Pour vérifier si le formulaire a été soumis et si les données retournées sont correctes, vous pouvez utiliser les méthodes `isSubmitted()` et `isValid()` de l'objet **$form**.
   - C'est à ce moment-là que vos contraintes de validation vont se mettre en action. En cas d'erreur, la condition ne sera pas acceptée et le formulaire affichera le ou les messages des erreurs concernées.
   - Voici comment vous pouvez le faire dans votre contrôleur Symfony :

```
public function nouvelEtudiant(Request $request)
    {
        $etudiant = new Etudiant();
```

apcpedagogie.com

```
        $form = $this->createForm(EtudiantType::class, $etudiant);
        $form->handleRequest($request);
        // Vérifier si le formulaire a été soumis et si les données sont
valides
        if ($form->isSubmitted() && $form->isValid()) {
            // Traitement des données lorsque le formulaire est soumis
et valide
            // Par exemple, enregistrement en base de données,
redirection, etc.
            return $this->redirectToRoute('nom_de_la_route');
        }
        // Afficher le formulaire dans le cas où il n'a pas été soumis
ou si les données ne sont pas valides
        return $this->render('etudiant/nouvel_etudiant.html.twig',
['formNouvelEtudiant' => $form->createView()]);
    }
```

5. **Traitement des données lorsque le formulaire est soumis et valide**

```
        if ($form->isSubmitted() && $form->isValid()) {
            $entityManager = $this->registry->getManager();
            $entityManager->persist($etudiant);
            $entityManager->flush();
            return $this->redirectToRoute('nom_de_la_route');
        }
```

6. **Assurez-vous que votre route est correctement définie dans le fichier `routes.yaml`:**

Sommaire du cours Symfony

Source : https://www.apcpedagogie.com/formulaire-symfony-par-la-pratique · téléchargé le 05/11/2026 · © APC Pédagogie · apcpedagogie.com