Développement web côté serveur avec Symfony

Module 06 - Formulaires

Objectifs

- Comprendre les bénéfices des formulaires Symfony
- Savoir afficher, valider et traiter les formulaires

Présentation des formulaires

Les problèmes des formulaires classiques

- Codes répétitifs
- Validation difficile
- HTML illisible
- Beaucoup d'erreurs possibles

Présentation des formulaires

Le composant Form de Symfony

- Automatise les tâches
- Validation très simple
- HTML ultra-limpide

Présentation des formulaires

Concepts de base

- Formulaire ⇔ Entité
- Sur soumission : données insérées dans l'entité automatiquement
- L'affichage est automatisé
- La validation se configure plutôt qu’elle ne se code
- Les classes sont réutilisables

Classe de formulaire

Créer une classe de formulaire

1. Avec `symfony console make:form`
2. Associer le formulaire à la classe
3. Adapter la classe générée

Formulaires

Classe de formulaire

Les types de champs

- Plus d'une trentaine de types de champs
- Les plus courants :
  - Text
  - TextArea
  - Email
  - Password
  - Integer
  - Search
  - Tel
  - Choice
  - Entity
  - Date
  - File

Classe de formulaire

Les options communes et essentielles

- label
- attr
- required

Démonstration

Affichage d’un formulaire : les étapes

1. Définir la classe de formulaire
2. Dans un contrôleur :
    1. Créer une instance de l'entité
    2. Créer une instance du formulaire
    3. Passer le formulaire à Twig
3. Déclencher l'affichage dans Twig

Affichage d'un formulaire : utiliser le formulaire dans le contrôleur

```php
class ProductController extends AbstractController
{
    /**
     * @Route("/produit/nouveau", name="product_create")
     */
    public function create(): Response
    {
        $product = new Product();
        $productForm = $this->createForm(ProductType::class, $product);
        return $this->render('product/create.html.twig', [
            "productForm" => $productForm->createView()
        ]);
    }
```

Affichage d'un formulaire sous Twig

```twig
<div>
    <h2>Ajouter un produit</h2>

    {# affichage en une ligne de code :) #}
    {{ form(productForm) }}
</div>
```

Formulaires

Affichage d'un formulaire sous Twig

```twig
<div>
    <h2>Ajouter un produit</h2>

    {# affichage avec légère décomposition #}
    {{ form_start(productForm) }}

    {{ form_widget(productForm) }}
    <button>Envoyer !</button>

    {{ form_end(productForm) }}
</div>
```

Formulaires

Affichage d'un formulaire sous Twig

```twig
<div>
    <h2>Ajouter un produit</h2>

    {# affichage avec décomposition par ligne #}
    {{ form_start(productForm) }}

    {{ form_row(productForm.name) }}
    {{ form_row(productForm.price) }}
    <button>Envoyer !</button>

    {{ form_end(productForm) }}
</div>
```

Formulaires

Affichage d'un formulaire sous Twig

```twig
<div>
    <h2>Ajouter un produit</h2>

    {# affichage en état de décomposition avancée #}
    {{ form_start(productForm) }}

    <div class="group">
        {{ form_label(productForm.name) }}
        {{ form_widget(productForm.name) }}
        {{ form_errors(productForm.name) }}
    </div>

    <div class="group">
        {{ form_label(productForm.price) }}
        {{ form_widget(productForm.price) }}
        {{ form_errors(productForm.price) }}
    </div>

    <button>Envoyer !</button>
    {{ form_end(productForm) }}
</div>
```

Formulaires

Affichage d'un formulaire sous Twig

```twig
<div>
    <h2>Ajouter un produit</h2>

    {# décomposition maniaque, nouveau en SF5.2 #}
    {{ form_start(productForm) }}

    <div class="group">
        <label for="{{ field_name(productForm.name) }}">Nom du produit</label>
        <input type="text"
               id="{{ field_name(productForm.name) }}"
               name="{{ field_name(productForm.name) }}"
               value="{{ field_value(productForm.name) }}"
               placeholder="{{ field_label(productForm.name) }}"
               class="form-control"
        >

        {% for error in field_errors(productForm.name) %}
            <div class="error">{{ error }}</div>
        {% endfor %}
    </div>

    {# ... #}

    <button>Envoyer !</button>
    {{ form_end(productForm) }}
</div>
```

Formulaires

Personnalisation de l'affichage

- Décomposer l'affichage
- Utiliser les sélecteurs CSS !
- Ajouter des `class`, utiliser les `id`
- Les thèmes de formulaire fournis
  - Bootstrap
  - Foundation
- Créer un thème de formulaire

Démonstration

Traitement d’un formulaire

- Dans le contrôleur
- Sur la même page
- Objectifs habituels :
  1. Tester si le formulaire est soumis
  2. Si oui :
     - Récupérer les données
     - Faire quelque chose avec les données
     - Rediriger vers une autre page
     - Afficher un message à l'utilisateur
- Symfony injecte les données dans l’entité

Formulaires

Traitement d'un formulaire

```php
/**
 * @Route("/produit/nouveau", name="product_create")
 */
public function create(Request $request): Response
{
    $product = new Product();
    $productForm = $this->createForm(ProductType::class, $product);

    $productForm->handleRequest($request);

    if ($productForm->isSubmitted()){
        //faire quelque chose avec les données
        //dump($product);

        return $this->redirectToRoute('main_home');
    }

    return $this->render('product/create.html.twig', [
        "productForm" => $productForm->createView()
    ]);
}
```

Traitement d’un formulaire

Afficher un message sur la page suivante

- Messages Flash
- Stockés en session
- Détruits dès qu'ils sont affichés

Formulaires

Traitement d'un formulaire

Créer et afficher un message Flash

```php
if ($productForm->isSubmitted()){
    //faire quelque chose avec les données
    //dump($product);

    $this->addFlash('success', 'Produit créé !');

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

```twig
{# dans base.html.twig... #}
</header>

{% for label, messages in app.flashes %}
    {% for message in messages %}
        <div class="alert-{{ label }}">
            {{ message }}
        </div>
    {% endfor %}
{% endfor %}
```

Démonstration

Formulaires

Validation des données

- S'assurer que l'utilisateur envoie des données telles qu'attendues
- Très fastidieux à coder
- Sous Symfony, on valide l'entité
- Configuration possible en XML, YAML, PHP ou annotations

Formulaires

Contraintes de validation

- Contrainte == règle de validation
- Appliquer 0, 1 ou plusieurs contraintes par champ
- Plus de 40 contraintes de base existent
- Possibilité de créer nos propres contraintes
- Chaque contrainte a des options prédéterminées

Formulaires

Contraintes de validation

- Quelques contraintes utiles :
  - `NotBlank`
  - `Type`
  - `Email`
  - `Length`
  - `Url`
  - `Regex`
  - `Range`
  - `EqualTo / NotEqualTo`
  - `LessThan / GreaterThan`
  - `Date`
  - `Choice`
  - `UniqueEntity`
  - `File`
  - `Image`
  - `Callback`

Appliquer des contraintes de validation

- Dans l'entité
- En annotation
- Se référer à la documentation pour les options
- On donne un alias *Assert* à la classe *Constraints*

```php
//ne pas oublier ce use !
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Entity(repositoryClass=ProductRepository::class)
 */
class Product
{
```

Formulaires

Appliquer des contraintes de validation

```php
private $id;

/**
 * @Assert\NotBlank(message="Veuillez renseigner un nom de produit !")
 * @Assert\Length(min="2", max="50",
 *     minMessage="Trop court ! Au moins 2 caractères !",
 *     maxMessage="Trop long ! Maximum 50 caractères !",
 * )
 * @ORM\Column(type="string", length=50)
 */
private $name;

/**
 * @Assert\Type(type="integer", message="Nos prix sont en entiers seulement !")
 * @Assert\NotBlank(message="Veuillez renseigner le prix du produit !")
 * @Assert\Range(min="1", max="1000",
 *     minMessage="Prix minimum de 1€ !",
 *     maxMessage="Prix maximum de 1000€ !"
 * )
 * @ORM\Column(type="integer")
 */
private $price;
```

Formulaires

Déclencher la validation

- Dans le contrôleur :

```php
$productForm->handleRequest($request);

//ajout du isValid()
if ($productForm->isSubmitted() && $productForm->isValid()){
    //faire quelque chose avec les données
    //dump($product);
```

- Les messages d'erreur s'affichent automatiquement

Démonstration

Sécurité : les attaques CSRF

- Cross Site Request Forgery
- Lorsqu'on soumet sans le vouloir un formulaire vers un site sur lequel on est connecté
- Protection : ajouter un champ caché aux formulaires dont nous seuls connaissons la valeur
- Protection automatique par Symfony !

Démonstration

Conclusion

- Vous comprenez les avantages apportés par les formulaires Symfony
- Vous savez afficher, valider et traiter les formulaires