Créer un thème personnalisé

Personnaliser le style et l'affichage d'un formulaire sans thème

La première manière de personnaliser l'affichage de ses formulaires est de le faire simplement en `CSS` en ajoutant des classes sur les éléments de formulaire grâce à `Twig`.

Voici le code de la vidéo, en commençant par `templates/page1.html.twig` :

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

{% block content %}

  <style>
  .form-container {
    display: flex;
    flex-flow: column;
  }

  .form-row {
    display: flex;
    flex-flow: column;
    margin-bottom: 15px;
  }

  .form-row label {
    font-weight: 700;
    font-size: 14px;
    margin-bottom: 5px;
  }

  .form-row input {
    background: #eee;
    border: 0;
    border-radius: 3px;
    padding: 10px 15px;
  }

  .form-row select {
    padding: 10px 15px;
```

```
    }

    .form-row ul {
      list-style: none;
      padding-left: 0px;
      margin-top: 5px;
    }

    .form-row ul li {
      font-size: 12px;
      font-weight: 700;
      color: #c0392b;
    }

    .btn {
      padding: 10px 15px;
      color: white;
      border: 0;
      border-radius: 3px;
      background-color: #3498db;
    }
    </style>


    {{ form_start(myform) }}
    <div class="form-container">
      <div class="form-row">
      {{ form_label(myform.content) }}
      {{ form_widget(myform.content) }}
      {{ form_errors(myform.content) }}
      </div>
      <div class="form-row">
      {{ form_label(myform.country) }}
      {{ form_widget(myform.country) }}
      {{ form_errors(myform.country) }}
      </div>
      <button type="submit" class="btn btn-primary">Sauvegarder</button
>
    </div>
    {{ form_end(myform) }}

{% endblock %}
```

Voici le code du contrôleur dans `src/Controller/DefaultController.php` :

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\Extension\Core\Type\CountryType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Validator\Constraints\Length;
use Symfony\Component\Validator\Constraints\NotBlank;;

class DefaultController extends AbstractController
{
  function __construct()
  {
  }

  #[Route('/', name: 'index')]
  public function index(Request $request)
  {
    $todo = new Todo();
    $form = $this->createFormBuilder($todo)
      ->add('content', TextType::class, [
        'label' => 'Un super label',
        'attr' => [
          'placeholder' => 'Contenu de la todo'
        ],
        'help' => 'Indiquez ce que vous avez à faire',
        'constraints' => [
          new NotBlank(message: 'Le contenu ne doit pas être vide'),
          new Length([
            'min' => 10,
            'max' => 50,
            'minMessage' => 'Le contenu doit faire au moins {{ limit
}} caractères',
            'maxMessage' => 'Le contenu doit faire au plus {{ limit
}} caractères',
          ])
        ],
      ])
      ->add('country', CountryType::class)
      ->getForm();
```

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

        if ($form->isSubmitted() && $form->isValid()) {
        }

        return $this->render('page1.html.twig', [
          'myform' => $form->createView()
        ]);
    }
}

class Todo
{
    function __construct(
        public string $content = '',
        public ?string $country = null,
    ) {
    }
}
```

Voici enfin le style du fichier `public/css/style.css` pour centrer le formulaire :

```
body {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  margin: 0;
  box-sizing: content-box;
}

.main-container {
  flex: 1;
  display: flex;
  justify-content: center;
  align-items: center;
}

header {
  background-color: red;
}

footer {
  background-color: black;
```

```
  color: white;
}
```

Personnaliser le style et l'affichage d'un formulaire avec un thème personnalisé

Si vous avez beaucoup de formulaires dans votre application, il peut être intéressant de créer un thème.

Commencez par créer un fichier `form-theme.html.twig` dans le dossier `template`.

Ensuite, modifiez la configuration du fichier `config/packages/twig.yaml` pour utiliser ce `template` comme thème :

```
twig:
  default_path: '%kernel.project_dir%/templates'
  form_themes: ['form-theme.html.twig']

when@test:
  twig:
    strict_variables: true
```

Modifions ensuite simplement l'ordre d'affichage des erreurs et du `widget HTML`.

Nous modifions simplement le bloc concerné dans le thème préconfiguré par `Symfony` :

```
{% use 'form_div_layout.html.twig' %}

{%- block form_row -%}
  {%- set widget_attr = {} -%}
  {%- if help is not empty -%}
    {%- set widget_attr = {attr: {'aria-describedby': id ~"_help"}} -
%}
  {%- endif -%}
  <div{% with {attr: row_attr} %}{{ block('attributes') }}{% endwith
%}>
    {{- form_label(form) -}}
    {{- form_widget(form, widget_attr) -}}
    {{- form_errors(form) -}}
    {{- form_help(form) -}}
  </div>
```

```
{%- endblock form_row -%}
```

`{% use 'form_div_layout.html.twig' %}` permet d'utiliser le thème de base `Symfony` pour les formulaires et de l'étendre.

En effet, créer un thème entier `Symfony` est très long et laborieux car il faut prévoir tous les champs possibles. Il faut donc écrire plusieurs centaines de `template`.

Modifions enfin notre `template templates/page1.html.twig` :

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

{% block content %}

  <style>
  .form-container {
    display: flex;
    flex-flow: column;
  }

  .form-row {
    display: flex;
    flex-flow: column;
    margin-bottom: 15px;
  }

  .form-row label {
    font-weight: 700;
    font-size: 14px;
    margin-bottom: 5px;
  }

  .form-row input {
    background: #eee;
    border: 0;
    border-radius: 3px;
    padding: 10px 15px;
  }

  .form-row select {
    padding: 10px 15px;
  }
```

```
  .form-row ul {
    list-style: none;
    padding-left: 0px;
    margin-top: 5px;
  }

  .form-row ul li {
    font-size: 12px;
    font-weight: 700;
    color: #c0392b;
  }

  .btn {
    padding: 10px 15px;
    color: white;
    border: 0;
    border-radius: 3px;
    background-color: #3498db;
  }
  </style>


  {{ form_start(myform) }}
  <div class="form-container">
    {{ form_row(myform.content, {'row_attr': {'class': 'form-row'} })
}}
    {{ form_row(myform.country, {'row_attr': {'class': 'form-row'} })
}}
    <button type="submit" class="btn btn-primary">Sauvegarder</button
>
  </div>
  {{ form_end(myform) }}

{% endblock %}
```

Modifier certaines variables de formulaire dans les thèmes

Cette partie est vraiment avancée donc survolez là et revenez-y lorsque vous rencontrerez le problème.

Lorsque vous voulez ajouter à un thème, par exemple un attribut, sans supprimer les autres attributs du thème, il faut fusionner les attributs du thème avec le ou les attributs

que vous voulez ajouter.

Pour ce faire, il faut utiliser **le filtre** `Twig merge()` :

```
{%- set attr = attr|merge({'nouvel-attribut': 'valeur' }) -%}
```

`{%- -%}` permet d'évaluer une expression sans ajouter de ligne dans le `HTML` généré.

`set` permet simplement d'assigner une variable. Ici `attr` .

Ensuite, nous utilisons le filtre pour ajouter la nouvelle valeur à celles existantes.

Voici l'exemple de la vidéo, la modification du fichier `templates/form-theme.html.twig` :

```
{% use 'form_div_layout.html.twig' %}

{%- block form_row -%}
  {%- set widget_attr = {} -%}
  {%- if help is not empty -%}
    {%- set widget_attr = {attr: {'aria-describedby': id ~"_help"}} -
%}
  {%- endif -%}
  {%- set row_attr = row_attr|merge({class: row_attr.class|default
('') ~ ' theme-row-custom'}) -%}
  <div{% with {attr: row_attr} %}{{ block('attributes') }}{% endwith
%}>
    {{- form_label(form) -}}
    {{- form_widget(form, widget_attr) -}}
    {{- form_errors(form) -}}
    {{- form_help(form) -}}
  </div>
{%- endblock form_row -%}

{%- block form_errors -%}
  {%- if errors|length > 0 -%}
  <ul class="form-list-error">
    {%- for error in errors -%}
      <li>{{ error.message }}</li>
    {%- endfor -%}
  </ul>
  {%- endif -%}
{%- endblock form_errors -%}
```

`row_attr.class|default('')` : le filtre `default()` permet de préciser une valeur par défaut si `row_attr.class` est nul ou n'existe pas.

`~ ' theme-row-custom'` : permet de concaténer la classe précédente avec `theme-row-custom`. Notez bien l'espace avant `theme-row-custom` pour que la classe précédente ne soit pas collée avec la classe ajoutée.