

SYMFONY

framework PHP

Twig

Anthony PARIS - Formateur

LE MOTEUR TWIG

Symfony utilise un moteur de template qui s'appelle Twig.

L'intérêt d'utiliser un moteur de template c'est d'abord la clarté du code, on écrira pas du code PHP dans du HTML, mais aussi la rapidité de Twig, tout les templates sont compilés en PHP et mis en cache, seulement quand vous changez le code d'un template, Twig le recompile et le met encore en cache, ce qui est très pratique en production.

Pour utiliser Twig, il faut retenir ces trois syntaxes

`{{ ... }}` Afficher quelque chose, une variable par exemple

`{% ... %}` Faire quelque chose, définir une variable, une boucle, structure conditionnelles

`{# ... #}` Commenter du texte

Les templates peuvent (et doivent) être utilisés partout. Quand on enverra des e-mails, leurs contenus seront placés dans un template. Ainsi, en récupérant le contenu du template dans une variable quelconque, on pourra le passer à la fonction mail de notre choix.

Mais il en va de même pour un flux RSS par exemple ! Si l'on sait afficher une liste des news de notre site en HTML grâce au template `newsList.html.twig`, alors on saura afficher un fichier RSS en gardant le même contrôleur, mais en utilisant le template `newsList.rss.twig` à la place.

LE MOTEUR TWIG

Nos fichiers de templating se trouvent dans «templates/cms»

Pour le moment vous disposez de index.html.twig et base.html.twig

Nous reviendrons sur base.html.twig dans un second temps.

Renommez le dossier cms par blog et ouvrez index.html.twig

Supprimez le contenu complet et ajoutez simplement ceci :

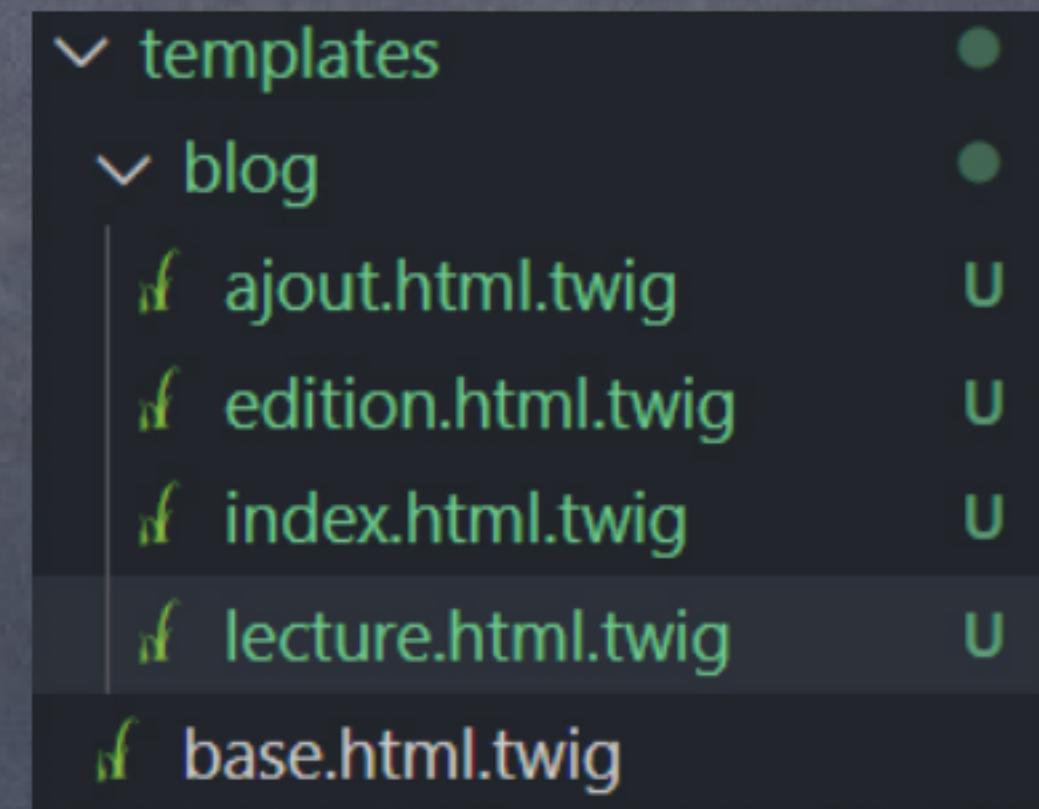
```
<h1>Bienvenu sur mon blog<h1>
```

Modifions BlogController pour y associer notre vue sur l'index, il existe plusieurs moyens, le plus rapide étant de passer par \$this->render (le raccourci)

```
public function index()
{
    return $this->render("blog/index.html.twig");
}
```

LE MOTEUR TWIG

Nous allons créer une vue par page sauf pour la suppression (pas besoin de page)



Nous allons modifier notre contrôleur pour rendre les vues et passer les variables si besoin

```
public function add()
{
    return $this->render("blog/ajout.html.twig");
}
```

```
public function edit($id)
{
    return $this->render("blog/edition.html.twig", ["id"=>$id]);
}
```

```
public function show($url,$id)
{
    return $this->render("blog/lecture.html.twig", ["url"=>$url,"id"=>$id]);
}
```

```
public function remove($id)
{
    return new Response('<h1>Supprimer l\'article ' . $id. '</h1>');
}
```

Le deuxième argument et le tableau de valeur à faire passer à la vue

LE MOTEUR TWIG

Nous allons modifier rapidement lecture pour afficher nos variables dans la vue



The screenshot shows a browser window with the address bar set to "sf4cours/article/test-5". The page content displays the rendered Twig template. It starts with an **h1** tag containing "Il s'agit de mon article". Below it is a **div** tag containing an **ul** list. The **ul** list contains two **li** items, each with a **strong** tag around the variable {{url}} and {{id}} respectively.

```
<h1>Il s'agit de mon article</h1>

<div>
    <ul>
        <li>URL : <strong>{{url}}</strong></li>
        <li>ID : <strong>{{id}}</strong></li>
    </ul>
</div>
```

Il s'agit de mon article

- URL : test
- ID : 5

Nous récupérons la variable et l'afficherons avec `{{VARIABLE}}` c'est plus propre qu'en PHP

VARIABLE TWIG

Voici un tableau récapitulatif de l'utilisation des variables avec twig

Afficher une variable	Pseudo : {{ pseudo }}	Pseudo : <?php echo \$pseudo; ?>
Afficher l'index d'un tableau	Identifiant : {{ user['id'] }}	Identifiant : <?php echo \$user['id']; ?>
Afficher l'attribut d'un objet, dont le getter respecte la convention \$objet->getAttribut()	Identifiant : {{ user.id }}	Identifiant : <?php echo \$user->getId(); ?>
Afficher une variable en lui appliquant un filtre. Ici, « upper » met tout en majuscules :	Pseudo en majuscules : {{ pseudo upper }}	Pseudo en lettre majuscules : <?php echo strtoupper(\$pseudo); ?>
Afficher une variable en combinant les filtres. « striptags » supprime les balises HTML. « title » met la première lettre de chaque mot en majuscule. Notez l'ordre d'application des filtres, ici striptags est appliqué, puis title.	Message : {{ news.texte striptags title }}	Message : <?php echo ucwords(strip_tags(\$news->getTexte())); ?>
Utiliser un filtre avec des arguments. Attention, il faut que date soit un objet de type Datetime ici.	Date : {{ date date('d/m/Y') }}	Date : <?php echo \$date->format('d/m/Y'); ?>
Concaténer	Identité : {{ nom ~ " " ~ prenom }}	Identité : <?php echo \$nom.' '.\$prenom; ?>

Pour {{obj.attrib}}

Elle vérifie si objet est un tableau, et si attribut en est un index valide. Si c'est le cas, elle affiche objet['attrib'].

Sinon, et si objet est un objet, elle vérifie si attribut en est un attribut valide (public donc). Si c'est le cas, elle affiche objet->attribut.

Sinon, et si objet est un objet, elle vérifie si attribut() en est une méthode valide (publique donc). Si c'est le cas, elle affiche objet->attribut().

Sinon, et si objet est un objet, elle vérifie si getAttribut() en est une méthode valide. Si c'est le cas, elle affiche objet->getAttribut().

Sinon, et si objet est un objet, elle vérifie si isAttribut() en est une méthode valide. Si c'est le cas, elle affiche objet->isAttribut().

Sinon, elle n'affiche rien et retourne null.

LES FILTRES TWIG

Twig dispose de twig prêt à l'emploi pour modifier l'affichage de vos variables

upper	Met toutes les lettres en majuscules.	<code>{{ var upper }}</code>
striptags	Supprime toutes les balises XML.	<code>{{ var striptags }}</code>
date	Formate la date selon le format donné en argument. La variable en entrée doit être une instance de <code>Datetime</code> .	<code>{{ date date('d/m/Y') }}</code> Date d'aujourd'hui : {{ "now" date('d/m/Y') }}
format	Insère des variables dans un texte, équivalent à <code>printf</code> .	<code>{{ "Il y a %s pommes et %s poires" format(153, nb_poires) }}</code>
length	Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne.	Longueur de la variable : <code>{{ texte length }}</code> Nombre d'éléments du tableau : {{ tableau length }}

Les filtres s'utilisent de cette façon :
`{{variable|filtre1|filtre2}}`

Il est possible de créer ses propres filtres.

Voici la documentation :
<https://twig.symfony.com/doc/3.x/filters/index.html>

SÉCURITÉ VARIABLES

Jusqu'à présent en php nous utilisions htmlspecialchars pour nettoyer et nous protéger des attaques, ici twig le fait automatiquement.

Ainsi, si le pseudo d'un de vos membres contient un «<» par exemple, lorsque vous écrivez {{ pseudo }} celui-ci est échappé, et le texte généré est en réalité «mon<pseudo» au lieu de «mon<pseudo», ce qui poserait problème dans votre structure HTML. Très pratique ! Et donc à savoir : inutile de protéger vos variables en amont, Twig s'occupe de tout en fin de chaîne !

Et dans le cas où vous voulez afficher volontairement une variable qui contient du HTML (JavaScript, etc.), et que vous ne voulez pas que Twig l'échappe, il vous faut utiliser le filtre raw comme ceci :{{ ma_variable_html|raw }}. Avec ce filtre, Twig désactive localement la protection HTML, et affiche la variable en brut, quel que soit ce qu'elle contient.

APP DANS TWIG

<code>{{ app.request }}</code>	La requête « request » qu'on a vue au chapitre précédent sur les contrôleurs.
<code>{{ app.session }}</code>	Le service « session » qu'on a vu également au chapitre précédent.
<code>{{ app.environment }}</code>	L'environnement courant : « dev », « prod », et ceux que vous avez définis.
<code>{{ app.debug }}</code>	<code>True</code> si le mode debug est activé, <code>False</code> sinon.
<code>{{ app.user }}</code>	L'utilisateur courant, que nous verrons également plus loin dans ce cours.
<code>{{ app.flashes('xxx') }}</code>	Récupère les messages flashes stockés dans la session.

On peut toutefois créer nos propres variables globales (plutôt constantes) de cette façon

```
# config/packages/twig.yaml  
# ...  
  
twig:  
    # ...  
    globals:  
        webmaster: "moi-même"
```

Ensuite dans votre vue :
`{{webmaster}}` affichera bien «moi-même»

LES BOUCLES ET CONDITIONS

if :

```
{% if membre.age < 12 %}  
    Il faut avoir au moins 12 ans pour ce film.  
{% elseif membre.age < 18 %}  
    OK bon film.  
{% else %}  
    Un peu vieux pour voir ce film non ?  
{% endif %}
```

```
<?php if($membre->getAge() < 12) { ?>  
    Il faut avoir au moins 12 ans pour ce film.  
<?php } elseif($membre->getAge() < 18) { ?>  
    OK bon film.  
<?php } else { ?>  
    Un peu vieux pour voir ce film non ?  
<?php } ?>
```

Tertiaire :

```
{{ foo ? 'yes' : 'no' }}  
{{ foo ?: 'no' }} is the same as {{ foo ? foo : 'no' }}  
{{ foo ? 'yes' }} is the same as {{ foo ? 'yes' : '' }}
```

null coalescence

```
# returns the value of foo if it is defined and not null, 'no' otherwise #  
{{ foo ?? 'no' }}
```

Avec twig pas de switch!!!

LES BOUCLES ET CONDITIONS

for :

```
<ul>
  {% for membre in liste_membres %}
    <li>{{ membre.pseudo }}</li>
  {% else %}
    <li>Pas d'utilisateur trouvé.</li>
  {% endfor %}
</ul>
```

```
<ul>
<?php if(count($liste_membres) > 0) {
  foreach($liste_membres as $membre) {
    echo '<li>' . $membre->getPseudo() . '</li>';
  }
} else { ?>
  <li>Pas d'utilisateur trouvé.</li>
<?php } ?>
</ul>
```

```
<select>
  {% for valeur, option in liste_options %}
    <option value="{{ valeur }}">{{ option }}</option>
  {% endfor %}
</select>
```

```
<?php
foreach($liste_options as $valeur => $option) {
  // ...
}
```

set : (définir une variable)

```
{% set foo = 'bar' %}
```

```
<?php $foo = 'bar'; ?>
```

LES BOUCLES ET CONDITIONS

Pour la boucle `for` il existe une variable «`loop`» qui permet d'intéragir avec la boucle

<code>{{ loop.index }}</code>	Le numéro de l'itération courante (en commençant par 1).
<code>{loop.index0}</code> }	Le numéro de l'itération courante (en commençant par 0).
<code>{loop.revindex}</code> }	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 1).
<code>{loop.revindex0}</code>	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 0).
<code>{loop.first}</code>	<code>true</code> si c'est la première itération, <code>false</code> sinon.
<code>{loop.last}</code>	<code>true</code> si c'est la dernière itération, <code>false</code> sinon.
<code>{loop.length}</code>	Le nombre total d'itérations dans la boucle.

TESTS DANS TWIG

Avec twig il est possible de tester ses variables, le test le plus utile :

```
{% if var is defined %} ... {% endif %}
```

php :

```
<?php if(isset($var)) { }>
```

Vous avez également «if var is null» pour tester si la variable existe mais = à null

<https://twig.symfony.com/doc/3.x/tests/index.html>

HERITAGE TEMPLATE

Quand on travaille sur un projet, généralement nous avons des blocs de nos pages qui se ressemblent, c'est le cas de la barre de navigation ou du pied de page.

Avec Twig, nous n'allons pas redéfinir ces blocs sur toutes les pages, nous allons définir un template mère qui va contenir l'architecture de base de nos pages et dont les pages de notre site vont hériter.

Le template mère va définir des blocs (block) dans lesquels on pourra rajouter du contenu spécifique pour chaque page. Cela fonctionne comme l'héritage en PHP, on peut redéfinir les variables et méthodes de la classe mère.

Dans notre cas, nous allons utiliser le fichier templates/base.html.twig comme template mère, il va donc contenir la barre de navigation de notre site et définir un bloc pour le contenu de nos pages.

HERITAGE TEMPLATE

Ce fichier définit le template de base de toutes les pages de notre site, toutes nos pages devront donc hériter de ce fichier. Nous définissons ici 4 blocs (`block`) à savoir `title`, `stylesheets`, `content` et `scripts`. Les blocs vont nous servir à quoi au juste? A pouvoir les remplacer dans les templates enfants

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Symfony Blog{% endblock %}</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    {% block stylesheets %}
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css" integrity="sha384-0evHe/X+R7YkIZDRvuzKMRqM-"

    {% endblock %}
</head>
<body>
    {{ render(controller("App\\Controller\\BlogController::menu")) }}

    {% block content %}

    {% endblock %}

    {% block scripts %}
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.5/dist/umd/popper.min.js" integrity="sha384-Xe+8cL9oJa6tN/veChSP7q+mnSPaj5Bcu9mPX5F5xIGE0I"
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/js/bootstrap.min.js" integrity="sha384-kjU+l4N0Yf4Z0JErLsIcvOU2qSb74wXpOhqTvwVx3OE"
    {% endblock %}
</body>
</html>| You, il y a 4 secondes • Uncommitted changes
```

HERITAGE TEMPLATE

On utilise extends comme en PHP aussi. Nous mettons juste le nom du fichier, le chemin de base pour tous les templates; c'est le dossier templates, le fichier base.html.twig se trouve directement dans ce dossier, pas dans un sous dossier, donc on ne spécifie aucun chemin ici.

Nous utilisons les même block pour surcharger le contenu par rapport au template défini «base» que l'on a hérité ici.

A noter nous pouvons utiliser {{parent()}} pour reprendre le contenu par défaut et le surcharger afin de ne rien écraser (utile pour les scripts par exemple)

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

% block title %}Ma page d'accueil% endblock %

% block content %
<div class="container">
    <div class="row">
        <div class="col ">
            <h1 class="text-center">Bienvenu sur mon blog</h1>
        </div>
    </div>
</div>
% endblock %
```

INCLUSION TWIG

`Extends` permet d'hériter une template au niveau de ses enfants (les fichiers possédant la balise `extends`)

Il peut arriver qu'une partie de template soit utile pour deux trois fichiers (un formulaire, une bannière, ...) dans ce cas nous utiliserons «`include`».

Créez un dossier `parts` dans `template` et un fichier `banniere.html.twig`

```
<div class="container-fluid">
    <div class="row">
        
    </div>
</div>
```

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

{% block title %}Ma page d'accueil{% endblock %}

{% block content %}
{% include "parts/banniere.html.twig" %}
<div class="container">
    <div class="row">
        <div class="col ">
            <h1 class="text-center">Bienvenu sur mon blog</h1>
        </div>
    </div>
</div>
{% endblock %}
```

← Dans `index.html.twig` on ajoute notre bannière par inclusion, cela nous permet de l'ajouter autant de fois que l'on souhaite dans une ou plusieurs pages
Si on a besoin d'envoyer des données à notre `include` :

```
{% include 'content.html.twig'
    with {'picture': object.picture, 'link': object.link,
    'name': object.name, 'info': object.info} %}
```

INCLUSION CONTRÔLEUR

Voici un dernier point à savoir absolument avec Twig, un des points les plus puissants dans son utilisation avec Symfony. On vient de voir comment inclure des templates : ceux-ci profitent des variables du template qui fait l'inclusion, très bien.

Pourtant dans certains cas, depuis le template qui fait l'inclusion, vous voudrez inclure un autre template, mais n'aurez pas les variables nécessaires pour lui. Restons sur l'exemple de notre plateforme d'annonces, dans le schéma précédent je vous ai mis un bloc rouge : considérons que dans cette partie du menu, accessible sur toutes les pages même hors de la liste des annonces, on veut afficher les 3 dernières annonces.

C'est donc depuis le layout général qu'on va inclure non pas un template – nous n'aurions pas les variables à lui donner –, mais un contrôleur. Le contrôleur va créer les variables dont il a besoin, et les donner à son template, pour ensuite être inclus là où on le veut !

Dans BlogController > On ajoute une méthode sans définir de routing, cela nous permettra d'ajouter notre menu ici en dur mais vous pourriez utiliser également votre BDD

```
public function menu()
{
    $listMenu = array(
        array('title' => 'Mon blog', 'texte'=>'Accueil', 'url'=>$this->generateUrl('homepage',[], UrlGeneratorInterface::ABSOLUTE_URL)),
        array('title' => 'Login', 'texte'=>'Connexion', 'url'=>"/login")
    );

    return $this->render("parts/menu.html.twig", array(
        'listMenu' => $listMenu
    ));
}
```

INCLUSION CONTRÔLEUR

Créez parts/menu.html.twig avec un for nous restituons notre tableau

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <a class="navbar-brand" href="/">Mon Site Internet</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse " id="navbarSupportedContent">
    <ul class="navbar-nav ml-auto mr-5">
      {% for menu in listMenu %}
        <li class="nav-item {% if loop.index == 1 %} active{% endif %}">
          <a class="nav-link" href="{{menu.url}}" title="{{menu.title}}>{{menu.texte}}</a>
        </li>
      {% endfor %}
    </ul>
  </div>
</nav>
```

INCLUSION CONTRÔLEUR

Dans notre fichier base.html.twig on ajoute le render controller, lancez maintenant votre fichier index

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Symfony Blog{% endblock %}</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    {% block stylesheets %}
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="
    {% endblock %}
</head>
<body>
    {{ render(controller("App\\Controller\\BlogController::menu")) }}

    {% block content %}
        You, a few seconds ago • Uncommitted changes
    {% endblock %}

    {% block scripts %}
        <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0
        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJ
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2
    {% endblock %}
</body>
</html>
```

RENDU CONTRÔLEUR

Nous avons notre menu avec nos liens, la bannière, et le contenu de index.html.twig avec inclusion de bootstrap et jquery

The screenshot shows a web page with a blue header bar. On the left of the header is the text "Mon Site Internet". On the right is a link to "Accueil" and "Connexion". The main content area has a large banner with the text "Mon blog est super". Below the banner, the text "Bienvenu sur mon blog" is displayed. At the bottom of the page, there is a footer bar with performance metrics: "200 @ homepage 406 ms 2.0 MiB 6 in 1.86 ms anon. 331 ms" and a "sf 4.4.8" logo.

CREATION DE NOS VUES

Dans base.html.twig nous allons ajouter du css inline (nous modifierons dans un fichier externe en suite)

```
{% block stylesheets %}
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"/>
    <style>
        /* article card */
        .article-card {
            margin-bottom: 30px;
        }

        .article-card h2 {
            font-size: 1.5rem;
            margin: 0;
            margin-top: 10px;
            margin-bottom: 5px;
        }

        .article-card h2 a {
            color: #d4b672;
        }

        .article-card.image-card {
            background-size: cover !important;
            background-position: center !important;
            background-repeat: no-repeat !important;
            height: 300px;
            position: relative;
            z-index: 1;
            color: #fff;
            display: flex;
            flex-direction: column-reverse;
            justify-content: flex-end;
            padding: 20px;
        }
    </style>
```

```
.article-card.image-card {
    background-size: cover !important;
    background-position: center !important;
    background-repeat: no-repeat !important;
    height: 300px;
    position: relative;
    z-index: 1;
    color: #fff;
    display: flex;
    flex-direction: column;
    justify-content: flex-end;
    padding: 20px;
}

.article-card.image-card::after {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, .7);
    z-index: -1;
}

.article-card.image-card h2 a {
    font-size: 2.2rem;
    color: #fff;
}

```

You, a few seconds ago • Uncommitted changes

</style>

justify-content ne fonctionne pas avec column-reserve

CREATION DE NOS VUES

Dans index.html.twig

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

{% block title %}Ma page d'accueil| {{ parent() }}{% endblock %}

{% block content %}
{% include "parts/banniere.html.twig" %}
<div class="container">
    <div class="row">
        <div class="col ">
            <h1 class="text-center">Bienvenu sur mon blog</h1>
        </div>
    </div>
</div>
<div class="container">
    <div class="row" style="height:300px;">
        <div class="col col-12 col-sm-7 col-md-9" style="color:white;">
            <div class="h-75 article-card image-card" style="background: url('https://images.pexels.com/photos/276452/pexels-photo-276452.jpeg?auto=compress&cs=tinysrgb&dpr=2&h=750&w=1260');">
                <h2 class="pl-2 align-text-bottom"><a href="" style="color:white">Titre</a></h2>
                <span class="date pl-2 align-text-bottom">14/01/2020</span>
            </div>
        </div>
        <div class="col col-12 col-sm-5 col-md-3" style="color:white">
            <div class="h-75 article-card image-card" style="background: url('https://images.pexels.com/photos/276452/pexels-photo-276452.jpeg?auto=compress&cs=tinysrgb&dpr=2&h=750&w=1260');">
                <h2 class="pl-2 align-text-bottom"><a href="" style="color:white">Titre</a></h2>
                <span class="date pl-2 align-text-bottom">14/01/2020</span>
            </div>
        </div>
    </div>
</div>
```

CREATION DE NOS VUES

Dans index.html.twig

```
<div class="container">
  <div class="row">
    <div class="col col-12 col-sm-6 col-md-3">
      <div class="article-card">
        
        <h2><a href="">Another Title</a></h2>
        <span class="date">14/01/2019</span>
      </div>
    </div>
    <div class="col col-12 col-sm-6 col-md-3">
      <div class="article-card">
        
        <h2><a href="">Another Title</a></h2>
        <span class="date">14/01/2019</span>
      </div>
    </div>
    <div class="col col-12 col-sm-6 col-md-3">
      <div class="article-card">
        
        <h2><a href="">Another Title</a></h2>
        <span class="date">14/01/2019</span>
      </div>
    </div>
    <div class="col col-12 col-sm-6 col-md-3">
      <div class="article-card">
        
        <h2><a href="">Another Title</a></h2>
        <span class="date">14/01/2019</span>
      </div>
    </div>
  </div>
</div>

{% endblock %}
```

CREATION DE NOS VUES

Rendu :

Mon Site Internet

Accueil Connexion

Mon blog est super

Bienvenu sur mon blog

Titre
14/01/2020

```
if (!extension_loaded("pcre")) {  
    die("PHP 5.2 or greater is required.  
    phpSysInfo requires the pcre extension to php in order to work properly.");  
}  
require_once APP_ROOT . '/includes/autoload.inc.php';  
// Load configuration  
require_once APP_ROOT . '/config.php';  
if (!defined('PSI_DEBUG')) {  
    require_once APP_ROOT . '/html/error_config.html';  
} else {  
    require_once APP_ROOT . '/html/error_config.php';  
}
```

Titre
14/01/2020

```
if (!extension_loaded("pcre")) {  
    die("PHP 5.2 or greater is required.  
    phpSysInfo requires the pcre extension to php in order to work properly.");  
}  
require_once APP_ROOT . '/includes/autoload.inc.php';  
// Load configuration  
require_once APP_ROOT . '/config.php';  
if (!defined('PSI_DEBUG')) {  
    require_once APP_ROOT . '/html/error_config.html';  
} else {  
    require_once APP_ROOT . '/html/error_config.php';  
}
```

Another Title
14/01/2020

```
if (!extension_loaded("pcre")) {  
    die("PHP 5.2 or greater is required.  
    phpSysInfo requires the pcre extension to php in order to work properly.");  
}  
require_once APP_ROOT . '/includes/autoload.inc.php';  
// Load configuration  
require_once APP_ROOT . '/config.php';  
if (!defined('PSI_DEBUG')) {  
    require_once APP_ROOT . '/html/error_config.html';  
} else {  
    require_once APP_ROOT . '/html/error_config.php';  
}
```

Another Title
14/01/2020

```
if (!extension_loaded("pcre")) {  
    die("PHP 5.2 or greater is required.  
    phpSysInfo requires the pcre extension to php in order to work properly.");  
}  
require_once APP_ROOT . '/includes/autoload.inc.php';  
// Load configuration  
require_once APP_ROOT . '/config.php';  
if (!defined('PSI_DEBUG')) {  
    require_once APP_ROOT . '/html/error_config.html';  
} else {  
    require_once APP_ROOT . '/html/error_config.php';  
}
```

Another Title
14/01/2020

```
if (!extension_loaded("pcre")) {  
    die("PHP 5.2 or greater is required.  
    phpSysInfo requires the pcre extension to php in order to work properly.");  
}  
require_once APP_ROOT . '/includes/autoload.inc.php';  
// Load configuration  
require_once APP_ROOT . '/config.php';  
if (!defined('PSI_DEBUG')) {  
    require_once APP_ROOT . '/html/error_config.html';  
} else {  
    require_once APP_ROOT . '/html/error_config.php';  
}
```

Another Title
14/01/2020

```
if (!extension_loaded("pcre")) {  
    die("PHP 5.2 or greater is required.  
    phpSysInfo requires the pcre extension to php in order to work properly.");  
}  
require_once APP_ROOT . '/includes/autoload.inc.php';  
// Load configuration  
require_once APP_ROOT . '/config.php';  
if (!defined('PSI_DEBUG')) {  
    require_once APP_ROOT . '/html/error_config.html';  
} else {  
    require_once APP_ROOT . '/html/error_config.php';  
}
```

CREATION DE NOS VUES

lecture.html.twig

```
{# templates/blog/show.html.twig #}
{% extends "base.html.twig" %}

{% block title %}{{ url}} | {{parent()}}{% endblock %}

{% block content %}
<article class="container">
    <div class="row">
        <div class="col col-12">
            <h2 class="text-center">{{ url|upper|replace({'-': ' '}) }}</h2>
            <span class="text-center d-block">14/01/2019</span>
            <p>
                
            </p>
            <span class="categories text-center d-block font-weight-bold text-uppercase">Catégories : php, html, css</span>
        </div>
    </div>
    <div class="row">
        <div class="col col-12">
            <p>
                Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
                tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
                quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
                consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
                cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
                proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
            </p>
            <p>
                Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
                tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
                quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
                consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
                cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
                proident, sunt in culpa qui officia deserunt mollit anim id est laborum. <br>
                Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
            </p>
        </div>
    </div>
</article>
{% endblock %}
```

CREATION DE NOS VUES

Dans l'ajout et la modification le formulaire sera identique en terme de champs donc on va utiliser le système d'inclusion de template. Créez un fichier parts/form_article.html.twig

```
<div class="container">
  <form>
    <div class="form-group row">
      <label for="image" class="col-sm-4 col-form-label">Image :</label>
      <div class="col-sm-8">
        <input type="file" class="form-control-file" name="image" id="image" placeholder="Votre image" aria-describedby="fileHelpId">
        <small id="fileHelpId" class="form-text text-muted">Chargement de votre image</small>
      </div>
    </div>
    <div class="form-group row">
      <label for="titre" class="col-sm-4 col-form-label">Titre de l'article</label>
      <div class="col-sm-8">
        <input type="text" class="form-control" name="titre" id="titre" placeholder="Titre de l'article">
      </div>
    </div>
    <div class="form-group row">
      <label for="categorie" class="col-sm-4 col-form-label">Catégories (php, html, css)</label>
      <div class="col-sm-8">
        <input type="text" class="form-control" name="categorie" id="categorie" placeholder="Catégories (php, html, css)">
      </div>
    </div>
    <div class="form-group row">
      <label for="contenu" class="col-sm-4 col-form-label">Intégrer le contenu de votre article</label>
      <textarea class="form-control" name="contenu" id="contenu" rows="6">Contenu de l'article</textarea>
    </div>
    <div class="form-check row">
      <label for="publication" class="form-check-label col-form-label">
        <input type="checkbox" class="form-check-input" name="publication" id="publication" value="1">
        Publier votre article?
      </label>
    </div>
    <div class="form-group row">
      <div class="offset-sm-2 col-sm-10">
        <button type="submit" class="btn btn-primary">Enregistrement de votre article</button>
      </div>
    </div>
  </form>
</div|
```

CREATION DE NOS VUES

Dans edition.html.twig

```
{# templates/blog/edit.html.twig #}
{% extends "base.html.twig" %}

{% block title %}Modifier{% endblock %}

{% block content %}
    <div class="container">
        <div class="row">
            <div class="col ">
                <h1 class="text-center">Modification article : {{id}}</h1>
                <hr>
            </div>
        </div>
    </div>
    {% include "parts/form_article.html.twig" %}
{% endblock %}
```

Dans ajout.html.twig

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

{% block title %}Ajouter un article{% endblock %}

{% block content %}
    <div class="container">
        <div class="row">
            <div class="col ">
                <h1 class="text-center">Ajouter un article</h1>
                <hr>
            </div>
        </div>
    </div>
    {% include "parts/form_article.html.twig" %}
{% endblock %}
```

Nous incluons le formulaire car il sera identique dans les deux fichiers de template.
L'inclusion de template est très utile pour éviter les redondances

CREATION DE NOS VUES

Rendu :

Ajouter un article

Non sécurisé | sf4cours/ajouter

Mon Site Internet

Ajouter un article

Image : Aucun fichier choisi
Chargement de votre image

Titre de l'article

Catégories (php, html, css)

Intégrez le contenu de votre article

Contenu de l'article

Publier votre article?

Enregistrement de votre article

AJOUTER NOS PROPRES PHOTOS ET CSS

Pour le moment nous avons ajouté du CSS inline, du CSS via CDN et des images distantes. Nous allons ajouter webpack encore à symfony

```
> composer require symfony/webpack-encore-bundle  
> yarn install
```

Cela va créer un webpack.config.jsfichier, ajouter le assets/répertoire et ajouter node_modules/à .gitignore.

Cette commande crée (ou modifie) un package.jsonfichier et télécharge les dépendances dans un node_modules/répertoire. Lors de l'utilisation de Yarn, un fichier appelé yarn.lock est également créé / mis à jour. Lorsque vous utilisez npm 5, un package-lock.jsonfichier est créé / mis à jour.

Symfony 6 fournit de nombreux changements qui peuvent perturber même les habitués du framework. La gestion des actifs en fait partie. Dans cet article, nous verrons comment gérer les ressources d'image avec Symfony 6.

Contrairement à Symfony 3, cette nouvelle version de Symfony utilise un nouveau système de gestion des actifs, le composant Webpack Encore .

Très simple d'utilisation, cette bibliothèque Javascript rend le travail avec CSS et JS très agréable.

CONFIGURATION WEBPACK ENCORE

Supposons que vous souhaitez stocker le CSS inline de votre site Web dans un fichier appelé `app.css`.

Dans une architecture de répertoire Symfony 6, vous placerez ce fichier sous le `assets/styles/app.css` dossier. (A réaliser)

Une fois terminé, ouvrez votre `webpack.config.js` fichier. Vous trouverez quelque chose comme ceci:

```
Encore
// directory where compiled assets will be stored
.set outputPath('public/build')
// public path used by the web server to access the output path
.set publicPath('/build')
// only needed for CDN's or sub-directory deploy
//.setManifestKeyPrefix('build')

/*
 * ENTRY CONFIG
 *
 * Add 1 entry for each "page" of your app
 * (including one that's included on every page - e.g. "app")
 *
 * Each entry will result in one JavaScript file (e.g. app.js)
 * and one CSS file (e.g. app.css) if your JavaScript imports CSS.
 */
.addEntry('assets/app', './assets/js/app.js')
//.addEntry('page1', './assets/js/page1.js')
//.addEntry('page2', './assets/js/page2.js')

// When enabled, Webpack "splits" your files into smaller pieces for greater optimization.
.splitEntryChunks()
```

Cela va créer en lançant la commande

`yarn encore dev (--watch)`

--watch modification en temps réel et compilation

un dossier `build` contenant vos fichiers css et js

Vous pouvez créer autant de points d'entrées que vous le souhaitez. On verra ensuite comment les utiliser.

Pour disposer de fichiers minifiés

`yarn encore production`

CONFIGURATION WEBPACK ENCORE

Dans base.html.twig ajoutez :

`encore_entry_link_tags` : restitue les fichiers CSS relatifs aux entrypoints définis

`encore_entry_script_tags` : restitue les fichiers JS relatifs aux entrypoints définis

Les fonctions `encore_entry_link_tags()` et `encore_entry_script_tags()` lisent un `entrypoints.json` fichier généré par Encore pour connaître le ou les noms de fichiers exacts à restituer.

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Symfony Blog{% endblock %}</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    {% block stylesheets %}
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css">
    {{ encore_entry_link_tags('app') }}
    {% endblock %}
</head>
<body>
    {{ render(controller("App\\Controller\\BlogController::menu")) }}

    {% block content %}

    {% endblock %}

    {% block scripts %}
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.5/dist/umd/popper.min.js">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/js/bootstrap.min.js">
    {{ encore_entry_script_tags('app') }}
    {% endblock %}
</body>
</html>      You, il y a 37 secondes • Uncommitted changes
```

CONFIGURATION WEBPACK ENCORE

La gestion des images est pratiquement la même, sauf que vous devrez rajouter une option dans le fichier de configuration de webpack.

```
.copyFiles(){  
  from: './assets/images',  
  // to path is relative to the build directory  
  to: 'images/[path][name].[ext]'  
}  
/*
```

Relancez la commande `yarn encore dev`

```
Error: Install file-loader to use copyFiles( )  
yarn add file-loader@^6.0.0 --dev
```

Et installer la dépendance demandée

CONFIGURATION WEBPACK ENCORE

Ajoutez un favicon dans le dossier «images/template» de «assets» et lancez la commande :

> yarn encore dev

Votre image sera ajoutée au build, pour l'ajouter sur votre site (dans base.html.twig)

```
<title>{% block title %}Symfony Blog{% endblock %}</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="icon" href="{{asset("images/template/favicon.ico")}}>
```

«asset» permet d'ajouter des ressources en se basant par rapport au dossier public en tant que racine

