

# Les "View" avec Twig

---

Symfony utilise par défaut un moteur de rendu appelé Twig. C'est un moteur de templates HTML qui permet d'intégrer facilement des variables et des structures de contrôle dans le HTML, sans écrire de PHP.

Bon à savoir : Twig peut aussi être utilisé sans Symfony, dans n'importe quel projet PHP.

Celui ci ajoute des fonctionnalités à nos fichiers html.

D'ailleurs vous noterez que nos fichiers sont nommés **nom.html.twig**.

Ces fichiers twig ne comporteront pas de PHP. Lorsque l'on va vouloir insérer des données, nous allons utiliser les fonctionnalités propres au twig.

## base.html.twig

Avant d'aller voir le fichier qui est appelé dans notre contrôleur, arrêtons nous sur **base.html.twig**.

Il débute comme un fichier html classique jusqu'à ce qu'on arrive à des accolades {}. Ces accolades sont les balises d'ouverture et de fermeture de twig, il y en a 3 types:

- {{ }} la double accolade correspond à un "**echo**" de PHP.
- {# #} correspond à un commentaire en twig.
- {% %} correspond à du code sans affichage.

## Blocks

Donc dans ce fichier base, nous trouvons le twig suivant :

- Dans la balise title nous avons du twig avec le mot-clef "**block**". Les blocks sont des zones de contenu que l'on nomme, et que l'on peut surcharger dans d'autres templates. Par défaut, un bloc affiche le contenu qu'il contient. Mais dès qu'un template enfant redéfinit ce bloc, le contenu initial est remplacé.
- Puis nous trouvons un block réservé pour les stylesheets (css) et un pour le javascript.
  - Ce dernier contient lui même un block réservé à "importmap" dont nous parlerons plus tard.
  - Ces blocks peuvent par exemple permettre d'ajouter ou remplacer les scripts et stylesheets qui pourraient différer dans une autre page.
- Et pour finir on trouve dans le body un block body actuellement vide.

Avant de quitter, changeons le title par défaut par "Cours Symfony".

## home/index.html.twig

Lorsque l'on crée un controller, symfony génère automatiquement une vue "index" correspondante dans un dossier du même nom que le contrôleur.

Voyons ce qui est écrit par défaut :

- On a le mot clef "**extends**" qui indique que ce fichier hérite de "base.html.twig". La majorité des templates d'une page contiendra cela. Important : Lorsqu'un template hérite d'un autre ({% extends "base.html.twig" %}), tout son contenu doit se trouver dans des blocs. Sinon, il sera ignoré.

- Puis on a le block title qui revient, si on regarde le title de notre page, il correspond à cette balise. Mais si on commente cette balise, on verra revenir le title par défaut que l'on avait défini. On a aussi la possibilité de ne pas effacer la valeur par défaut et juste ajouter la notre grâce à la fonction "**parent()**". Remplaçons le title actuel par :

```
{{ parent() }} - Accueil
```

Le titre est maintenant une fusion de celui par défaut et du nouveau.

- L'élément suivant est le block body. On le trouvait vide dans le fichier "Base" mais ici on a des éléments de base indiquant l'emplacement du controller et du template, on peut les supprimer. Tout ce qui est indiqué ici viendra remplacer le vide du fichier base.
- Il n'y a rien d'autre ici mais si on avait besoin d'un fichier css ou js spécifique à cette page, on pourrait appeler les blocks correspondants.
- Les noms des blocks sont totalement arbitraire, on peut les nommer comme on le souhaite et en créer autant qu'on veut. Si on avait besoin d'un footer qui va varier d'une page à l'autre, on pourrait très bien créer un block footer et l'appeler quand il doit changer.

## VS Code paramétrage

On notera que les **emmet** de HTML ne fonctionnent plus.

Car pour VS code on est dans un fichier en langage **twig** et non plus HTML. Ce n'est pas grave nous allons pouvoir modifier cela:

1. Rendons nous dans les paramètres vscode et cherchons **emmet.includeLanguages**
2. Ajoutons un élément.
3. En clef indiquons **twig**
4. En valeur indiquons **html**

Maintenant vscode comprend que lorsque l'on fait du twig, il faut qu'il continue à inclure les emmet de HTML.

## Affichage

Comme je le disais précédemment, la méthode render permet de fournir des variables à notre vue. Si j'écris :

```
{{ controller_name }}
```

On verra la valeur de la variable s'afficher.

Mais lorsque l'on affiche une variable, on peut aussi lui indiquer des filtres pour modifier la variable.

```
{{ controller_name | upper }}
```

La valeur de notre variable est maintenant en majuscule.

De nombreux filtres et fonctions sont utilisable, on pourra même créer les nôtres, en attendant, la liste complète se trouve ici :

<https://twig.symfony.com/doc/3.x/#reference>

Lorsque notre variable correspond à un tableau associatif ou à un objet, on peut accéder à la valeur directement avec :

"variable.propriété"

Par exemple si on prend notre variable "pays":

```
{{ pays.france }}  
<br>  
{{ pays.angleterre | reverse }}
```

ATTENTION: twig ne peut pas directement afficher un tableau complet.

## Conditions

Twig permet aussi de gérer les conditions if, elseif et else :

```
<strong>  
{% if chiffre > 5 %}  
    {{ chiffre }} est plus grand que 5  
{% elseif chiffre < 5 %}  
    {{ chiffre }} est plus petit que 5  
{% else %}  
    Votre chiffre vaut 5.  
{% endif %}  
</strong>
```

## Tableaux

Et évidemment on peut parcourir un tableau avec twig :

```
<ul>  
    {% for f in fruits %}  
        <li>{{ f }}</li>  
    {% endfor %}  
</ul>
```

Les filtres peuvent aussi être utilisé avec notre for, ajoutons par exemple "**| reverse**" après notre tableau. Le tableau sera alors inversé.

Une chose plaisante de twig c'est qu'il est possible de gérer les tableaux vide comme ceux qu'on obtient quand notre requête sql ne correspond à rien. Pour cela il nous suffit d'intégrer un "**else**" dans notre "**for**".

```
{% for v in vide %}
    Le tableau étant vide, cela ne s'affichera pas.
{% else %}
    Votre tableau est vide.
{% endfor %}
```

## XSS

Par défaut, la bibliothèque twig de symfony est paramétré pour filter toute les données affiché pour éviter les attaques xss:

```
{{xss}}
```

Si on veut ajouter du HTML ou du JS via une variable à afficher, il faudra ajouter le filtre `raw`;

## header et footer

Éloignons nous de ce fichier un instant pour créer dans nos templates un dossier "**layout**" contenant les fichiers "**\_header.html.twig**" et "**\_footer.html.twig**" et penchons nous sur le cas du premier.

Dans le h1 nous avons placé deux variables qui nous permettront de le changer, si on souhaite d'une page à l'autre un header différent. - header permettra de changer tout le contenu de notre header. - titlte permettra de changer juste le titre dans le h1.

```
<header>
    {% if header is defined %}
        {{header|raw}}
    {% else %}
        <h1>
            {{ title ?? "Cours Symfony" }}
        </h1>
    {% endif %}
</header>
```

Dans le footer nous retrouvons un lien vers notre page d'accueil. Une des difficultés lorsqu'on développe un site est de prévoir à l'avance comment seront construit toute les routes de notre site.

Si on change d'avis entre temps, cela nous oblige à parcourir tout notre site pour changer chaque liens. Ce n'est pas le cas avec Symfony et Twig.

```
<footer>
    <a href="">Retourner à l'accueil</a>
</footer>
```

## path()

Si vous vous souvenez, on a appelé la route vers notre page d'accueil "**app\_accueil**". Et bien je vais utiliser la fonction twig "path()" avec le nom de ma route en argument.

```
<a href="{{ path("app_accueil") }}">Retourner à l'accueil</a>
```

Cette fonction retournera le lien absolu correspondant à ce nom. Donc tant que je ne changerais pas de nom, la route peut changer, il sera automatiquement changé là où j'utilise cette fonction.

Notons que l'on peut aussi utiliser `url("nomDuChemin")` par exemple dans un email, qui retournera alors le lien comprenant le nom de domaine.

## include

Maintenant venons placer dans notre fichier "base.html.twig" les lignes suivante autour de notre block body:

```
{% include "layout/_header.html.twig" %}  
<main class="{{ mainClass ?? '' }}">  
    {% block body %}{% endblock %}  
</main>  
{% include "layout/_footer.html.twig" %}
```

Comme son nom l'indique, le mot clef **include** permettra d'inclure d'autres fichiers.

Ces fichiers twig ne comporteront pas de PHP. Lorsque l'on va vouloir insérer des données, nous allons utiliser les fonctionnalités propre au twig.

On notera que j'ai englobé nos pages dans une balise "main" qui a une classe optionnelle pouvant être ajouté directement dans nos pages en définissant la variable "mainClass".

## Inclure du CSS

Nous reparlerons du fonctionnement de CSS et JS avec symfony plus tard, Pour l'instant nous nous contenterons de copier coller le css inclu dans les ressources du cours dans le fichier suivant :

`/assets/styles/app.css`

## la variable global "app"

Cette variable globale est disponible n'importe où en twig.

Elle permettra de nombreuses choses comme :

- Récupérer les informations de l'utilisateur connecté.
- Récupérer les messages flash
- Récupérer les informations en session
- Récupérer les informations sur la requête actuelle

- Et ainsi de suite (voir la documentation officielle pour un détail complet)

Ici nous allons l'utiliser pour récupérer le nom de la route actuelle. Ajoutons à notre header ceci :

```
<nav class="header-navigation">
  <a href="{{path('app_accueil')}}" class="{{ app.current_route ==
'app_accueil'? 'active': '' }}">Accueil</a>
  <a href="" class="{{ app.current_route == 'app_user_list'? 'active': '' }}">Liste
Utilisateur</a>
</nav>
```

Maintenant si nous sommes sur une route nommée "app\_accueil" alors le lien correspondant gagnera une classe. De même pour le second lien avec "app\_user\_list" que nous construirons plus tard.

## Inclure un contrôleur

Imaginons le cas où nous voudrions que sur la droite de notre site, nous ayons la liste des articles récents. On ne va pas dans chacun de nos contrôleurs faire la requête pour obtenir ceux-ci.

Symfony et twig nous permettent d'inclure non plus une autre vue mais un contrôleur entier.

Pour cela il faut que la vue concernée ne contienne pas "**extends**" ou autre block mais seulement le contenu qui doit être affiché.

Ensuite nous allons nous placer là où on souhaite afficher notre élément et écrire :

```
{{ render(controller("App\\Controller\\NomDuController::NomDeLaMéthode", {les
possibles options ici})) }}
```