

# Les "View" avec Twig

---

Symfony utilise par défaut un moteur de rendu nommé **TWIG**.

Celui ci ajoute des fonctionnalités à nos fichiers html.

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

Ces fichiers twig ne comporterons pas de PHP. Lorsque l'on va vouloir insérer des données, nous allons utiliser les fonctionnalités propre 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 parties de code que l'on va nommer (ici le block est nommé "title") et qui se contentera d'afficher normalement le html par défaut qu'il contient.  
Cependant si dans une autre page, j'appelle ce même block, alors le contenu par défaut sera remplacé par le nouveau.
- Ensuite nous avons un commentaire qui indique que l'on peut installer "webpack-encore-bundle" si on souhaite utiliser "symfony UX". (nous en reparlerons plus tard)
- Puis nous avons un block stylesheet et un block javascript, parfois pour placer notre css et notre js. et ainsi le remplacer si certaines pages ont besoin de fichier différents.  
Actuellement on y trouve des fonctions liées à "webpack-encore-bundle" et "Symfony UX".
- 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".

- 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 en le remplissant ici, il va venir 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 correspondant.
- Les noms des blocks sont totalement arbitraire, on peut les nommer comme on le souhaite et en créer autant qu'on veut. Si on a 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.

Par défaut le block body est rempli d'un message rappelant où se trouve la vue (template) et le controller. On peut l'effacer.

## Affichage

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

```
{{ controller_name }}
```

On verra le nom 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 utilisables, 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 }}
```

## 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 %}
```

## 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 une variable qui nous permettra de le changer si on le souhaite d'une page à l'autre.

```
<header>
  <h1>
    {% if header %}
      {{header}}
    {% else %}
      Cours Symfony
    {% endif %}
  </h1>
</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 toutes les routes de notre site.

Si on change d'avis entre temps, cela nous oblige à parcourir tout notre site pour changer chaque lien. 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 "**Accueil**". Et bien je vais utiliser la fonction twig "path()" avec le nom de ma route en argument.

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

Cette fonction retournera le lien 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.

## include

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

```
{% include "layout/_header.html.twig" with {header : block("header")??""}%}
{% block body %}{% endblock %}
{% include "layout/_footer.html.twig" %}
```

Comme son nom l'indique, le mot clef **include** permettra d'inclure d'autres fichiers. Ici j'indique de donner à mon fichier header une variable nommée "**header**" qui contiendra le block header si il est défini sinon rien.

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

## 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 concerné ne contiennent 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})) }}
```