Symfony - Contrôleurs et Templating

Qu'est-ce que les contrôleurs

Symfony est un framework se basant sur le modèle MVC:

- Les **Entity** sont le modèle, la représentation des données de l'application.
- Les **Controller** sont la liaison entre les **Entity** et les vues.

Mais les **vues** dans tout ça ? Elles sont définies dans le dossier **templates**. Et elles sont écrites en **Twig**.

Qu'est-ce que Twig ?

Twig est un langage de **templating**. Le **templating** est une méthode de génération de documents à partir d'un modèle précis. Si ça parait flou ne vous inquiétez pas, nous verrons comment ça se concrétise juste après.

Le plus gros intérêt d'un système de templating comme Twig, c'est que vous pouvez dire adieu à tout le PHP dans vos vues ! Ainsi, le code suivant en PHP:

```
<body>
    <table>
        <?php
        $tab = [["name" => "Foo"], ["name" => "Bar"]];
        foreach ($tab as $val) {
        ?>
            <tr><td>
                    <?php
                      echo $val["name"];
                    ?>
            </td></tr>
        <?php } ?>
    </table>
</body>
```

Se transformera en

```
<body>
    <table>
        {% for val in tab %}
          <td> <tr>{{ val.name }} </tr> </td>
        {% endfor %}
    </table>
</body>
```

Exit donc le PHP et ses ouvertures de script juste pour une accolade !

Comment ça se concrétise dans le code ?

Twig servant à créer des vues, il est intrinsèquement lié au concept de contrôleur. On va donc pas pouvoir y couper.

Création d'un contrôleur

Allez dans le dossier avec un terminal, et rentrez la commande suivante:

```
symfony console make:controller ListUsersController
```

La commande va créer **deux fichiers**:

- **Controller/ListUsersController.php**: Notre fameux contrôleur
- **templates/list_users/index.html.twig**: La **vue** associée au contrôleur

Par défaut chaque contrôleur est associé à une vue. Dans certains cas cette vue ne sert pas, mais ce sont des utilisations plus avancées. Ici, on ne verra qu'un usage basique des contrôleurs Symfony, à savoir s'en servir pour **rendre** une vue.

Voyons le résultat

En étant dans le dossier du projet, exécutez dans votre terminal

```
symfony server:start -d
```

Ouvrez ensuite votre navigateur préféré et essayez d'accéder à **http://localhost:8000/list/users**. Vous devriez tomber sur:

Hello ListUsersController! ✅

This friendly message is coming from:

- Your controller at `src/Controller/ListUsersController.php`
- Your template at `templates/list_users/index.html.twig`

Figure 7: Notre magnifique contrôleur.

Décortiquons le contrôleur

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController; // 1
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ListUsersController extends AbstractController // 2
{
    #[Route('/list/users', name: 'app_list_users')] // 3
    public function index(): Response // 4
    {
        return \$this->render('list_users/index.html.twig', [ // 5
            'controller_name' => 'ListUsersController', // 6
        ]);
    }
}
```

Ca en fait beaucoup d'un coup, n'est-ce pas ? On va voir ça !

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController; // 1
```

- 1: Les **use** servent à **importer** un symbole. Ce symbole peut être aussi bien une variable qu'une fonction ou encore une **classe**. Avec Symfony, ce sera le plus souvent une **classe**.

```
class ListUsersController extends AbstractController // 2
```

- 2: Symfony, comme beaucoup de frameworks MVC, utilise la programmation **Orientée Objet**. Un contrôleur est une simple classe fille d'**AbstractController**.

```
#[Route('/list/users', name: 'app_list_users')] // 3
```

- 3: Cette ligne définit la **route** sur laquelle écoute notre contrôleur. Elle indique aussi à Symfony que c'est la fonction juste en-dessous qu'il faut appeller pour obtenir la **réponse** du contrôleur, aka ce que notre navigateur web recevra **en réponse**. Ici, l'URL à rentrer pour appeller ce contrôleur depuis notre navigateur sera donc localhost/**list/users**. Changeons-le pour **listusers**. Pour cela, la ligne 11 doit devenir:

```
    #[Route('/listusers', name: 'app_list_users')]
```

Ainsi, pour accéder à notre page, il suffira désormais de rentrer localhost/**listusers**.

```
public function index(): Response // 4
```

- 4: La déclaration de la fonction. Ici, Symfony la nomme par défaut **index**, mais ça pourrait aussi bien être **shrek_5**. Ca fonctionnerait tout autant. Vous pouvez d'ailleurs essayer. Le tout est d'avoir l'attribut **#[Route]** au-dessus de la déclaration de fonction.

```
return \$this→render('list_users/index.html.twig', ...) // 5
```

- 5: Dernière ligne mais pas des moindre, elle demande à Symfony de faire le **rendu** de la **vue**, avec les données associées. Cette demande s'effectue via l'appel de la méthode **render**, possible via

```
$this→render( ... )
```

Symfony va donc chercher le fichier **list_users/index.html.twig** puis, à la suite d'un processus fastidieux de **parsing** puis d'**instanciation de template** dont on a pas à se soucier, va renvoyer du HTML tout propre, que le client pourra voir en toute transparence.Ici, à la fonction est aussi passée le tableau associatif suivant:

```
['controller_name' ⇒ 'ListUsersController']
```

```
'controller_name' => 'ListUsersController' // 6
```

- 6: Cette ligne n'est pas des moindres, elle indique à Symfony de **substituer** au nom de variable **controller_name** la valeur **ListUsersController**. Vous pouvez d'ailleurs remplacer le texte 'ListUsersController' par ce que vous souhaitez et recharger la page pour observer un changement immédiat dans la page que vous recevrez. Vous pouvez aussi remplacer la clé, 'controller_name' par ce que vous voulez et recharger pour observer tout aussi rapidement que Symfony vous indiquera tout en politesse que vous avez foiré. Il vous fournira d'ailleurs un message d'erreur très détaillé. Quelle bonne poire ce Symfony !

Symfony Exception

Symfony Docs

RuntimeError

HTTP 500 Internal Server Error

Variable "controller_name" does not exist.

Exception | Logs 1 | Stack Trace

Twig\Error\
RuntimeError

▸ Show exception properties

in templates/list_users/**index.html.twig** (line 12)

```
7.      .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
8.      .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
9.  </style>
10.
11. <div class="example-wrapper">
12.     <h1>Hello {{ controller_name }}! ✅</h1>
13.
14.     This friendly message is coming from:
15.     <ul>
16.         <li>Your controller at <code><a href="{{ '/home/quentin/b2_symfony/src/Controller/ListUsersController.php'|file_link(0) }}">src
17.         <li>Your template at <code><a href="{{ '/home/quentin/b2_symfony/templates/list_users/index.html.twig'|file_link(0) }}">templat
```

in var/cache/dev/twig/f3/**f372e31cb0af9ba82d69dc4897859f89.php** -> {closure} (line 96)

in vendor/twig/twig/src/**Template.php** -> block_body (line 171)

in var/cache/dev/twig/21/**216f751894fa72ea9a62a82d96b9dd33.php** -> displayBlock (line 69)

in vendor/twig/twig/src/**Template.php** -> doDisplay (line 394)

in vendor/twig/twig/src/**Template.php** -> displayWithErrorHandling (line 367)

500 → @app_list_users 92 ms 8.0 MiB 1 n/a 0 ms Server 6.3.9

Figure 16: This is fine.

En résumé:

- Symfony fait un usage intensif de **classes**. Nous travaillons donc en **Programmation Orientée Objet**.
- L'attribut **Route** sert à définir un nouveau chemin dans notre site, dont la ressource renvoyée sera la valeur de retour de la fonction juste en-dessous, qui sera du type **Response**.
- **Instancier** un template Twig se fait donc via la méthode **render** accessible directement via le contrôleur. On peut aussi lui passer toutes les valeurs dont il a besoin pour instancier notre template.
- Symfony, dans son extrême bonté, nous fournira des messages d'erreur **très détaillés** en cas d'étourderie.

Rendons visite au fichier Twig

En ouvrant le fichier **templates/list_users/index.html.twig**, on se rend compte de l'ampleur du bordel:

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

{% block title %}Hello ListUsersController!{% endblock %}

{% block body %}
<style>
    .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font:
18px/1.5 sans-serif; }
    .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
    <h1>Hello {{ controller_name }}!    </h1>

    This friendly message is coming from:
    <ul>
        <li>Your controller at <code><a href="{{ '/home/quentin/b2_symfony/
src/Controller/ListUsersController.php'|file_link(0) }}">src/Controller/
ListUsersController.php</a></code></li>
        <li>Your template at <code><a href="{{ '/home/quentin/b2_symfony/
```

```
templates/list_users/index.html.twig'|file_link(0) }}">templates/list_users/
index.html.twig</a></code></li>
    </ul>
</div>
{% endblock %}
```

Avant que vos cerveaux se liquéfient on va supprimer ce dont on se fout, puis on va annoter le reste comme précédemment:

```
{% extends 'base.html.twig' %} <!-- 1 -->

{% block title %} <!-- 2 -->
  Hello ListUsersController!
{% endblock %} <!-- 3 -->

{% block body %} <!-- 4 -->
   ...
  <h1>Hello
    {{ controller_name }}! <!-- 5 -->
  </h1>
   ...
{% endblock %} <!-- 5 -->
```

Ne t'inquiète pas ça va bien spasser

Ici j'ai tronqué tout ce dont on se fout et j'ai mis des … ici car c'est soit des choses que vous connaissez (ou êtes censés déjà connaitre), soit des choses inutiles pour le moment. Bref:

```
{% extends 'base.html.twig' %} <!-- 1 -->
```

- 1: Un template peut lui-même être une **extension** d'un autre template. Je ne vais pas trop pousser dans les détails ici mais vous pouvez, en modifiant le fichier **templates/base.html.twig**, définir une **base** pour toutes vos pages sans avoir à copier-coller. Exit le copier-coller du header, footer et autres pages CSS. De quoi ravir nos crack-addicts à Tailwind :^).

```
{% block title %} <!-- 2 -->
```

- 2: Indique ce que Symfony doit mettre en titre de la page. Le bloc **title** est défini dans **base.html.twig** sous la forme suivante:

```
<title>
{% block title %}
  Welcome!
{% endblock %}
</title>
```

Ainsi, la valeur par défaut sera **Welcome!** mais vous pouvez l'écraser dans votre propre template, comme l'a fait Symfony pour nous !

```
{% endblock %} <!-- 3 -->
```

- 3: Marque la fin du bloc ouvert **au plus tôt**. Les blocs Twig suivent la même logique que les balises HTML au niveau de l'ordre de fermeture.

```
{% block body %} <!-- 4 -->
```

- 4: Exactement comme pour **title**, mais avec le body. Tout ce qu'on met dans le bloc sera inséré entre deux balises **body**.

```
<h1>Hello
    {{ controller_name }}! <!-- 5 -->
</h1>
```

- 5: Le plus intéressant ! Comme je l'ai précisé (beaucoup) plus haut, dans l'appel à la fonction **$this->render(...)**, nous pouvons passer des noms de variables sous forme de tableau associatif. Ici, {{ controller_name }} signifie **cherche la clé *controller_name* et remplace-moi par sa valeur**. Ainsi, avec le tableau associatif suivant:

```
  ['controller_name' => 'ListUsersController']
```

{{ controller_name }} sera remplacé par **ListUsersController**, ce qui explique mieux ce que l'on a reçu auparavant !

Hello ListUsersController! ✅

This friendly message is coming from:

- Your controller at `src/Controller/ListUsersController.php`
- Your template at `templates/list_users/index.html.twig`

Figure 28: Flashback.

Comme toutes les blagues, les templates Twig les plus courts sont les meilleurs. C'est ici la fin de notre template.

En résumé:

- Tout template peut **hériter** d'un autre, ce qui permet d'éviter **beaucoup** de répétition de code (exemple du **header**, **footer**, **link du css** et autres...).
- Le système de **bloc** permet d'éditer le contenu par défaut du template dont on hérite.
- Twig dispose aussi d'un système de variables. Toute variable utilisée dans le template doit être accompagnée d'une assignation au niveau du contrôleur associé. Sinon, c'est l'engueulade assurée.

Symfony Exception

Symfony Docs

RuntimeError

HTTP 500 Internal Server Error

Variable "controller_name" does not exist.

Exception | Logs 1 | Stack Trace

Twig\Error\
RuntimeError

▸ Show exception properties

in templates/list_users/**index.html.twig** (line 12)

```
7.      .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
8.      .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
9.  </style>
10.
11. <div class="example-wrapper">
12.     <h1>Hello {{ controller_name }}! ✅ </h1>
13.
14.     This friendly message is coming from:
15.     <ul>
16.         <li>Your controller at <code><a href="{{ '/home/quentin/b2_symfony/src/Controller/ListUsersController.php'|file_link(0) }}">src
17.         <li>Your template at <code><a href="{{ '/home/quentin/b2_symfony/templates/list_users/index.html.twig'|file_link(0) }}">templat
```

in var/cache/dev/twig/f3/**f372e31cb0af9ba82d69dc4897859f89.php** -> {closure} (line 96)

in vendor/twig/twig/src/**Template.php** -> block_body (line 171)

in var/cache/dev/twig/21/**216f751894fa72ea9a62a82d96b9dd33.php** -> displayBlock (line 69)

in vendor/twig/twig/src/**Template.php** -> doDisplay (line 394)

in vendor/twig/twig/src/**Template.php** -> displayWithErrorHandling (line 367)

500 → @app_list_users 92 ms 8.0 MiB 1 n/a 0 ms Server 6.3.9

Figure 30: Tout est lié :o

Sources

- Twig - Documentation
- Symfony - Creating and Using Templates