Module 08 - Templating avec Twig

Dans ce module, nous allons aborder le composant *Vue* du pattern MVC avec le moteur de template Twig.

Templates

Le principe d'un template est d'élaborer des modèles statiques de pages HTML dans lesquels viendront s'insérer des données dynamiques, résultat de l'exécution d'un programme.

Le *templating* permet de faire **la séparation entre données statiques (présentation) et données dynamiques (résultat du traitement)**.

> *N'est-ce pas une définition possible de PHP et l'origine de son succès ?!*

Twig

Twig (3.*) est un moteur de templates *fast* (compilé), *secure* (échappement par défaut) and *flexible* (on va voir pourquoi).

- Crée en 2008 ;
- Inspiré du moteur de templates Jinja(Python) ;
- Maintenu par la *core team* de Symfony, moteur de templates par défaut de Symfony ;
- **Utilisable dans n'importe quel projet**, comme n'importe quelle dépendance (composant à installer via Composer)

```
composer require "twig/twig:^3.0"
```

Pourquoi utiliser Twig et non PHP directement ?

PHP est **par définition** un moteur de templates. Alors, pourquoi ne pas utiliser directement PHP ?

Fabien Potencier, créateur (entre autres) du framework Symfony, a écrit un long billet intéressant à ce sujet.

Inconvénients d'utiliser *vanilla* PHP comme moteur de template :

- Trop verbeux ;
- Mauvaise lisibilité, car langage non orienté *template* ;
- Mauvaise réutilisabilité : pas de mécanisme de **réutilisation** de **fragments** de template (composants) ;
- **Sécurité : Pas d'échappement par défaut !**

Exemple Twig vs PHP : échappement des caractères spéciaux

En PHP, pour *échapper* les caractères spéciaux (et dangereux dans le contexte d'une page web !), il faut utiliser la fonction htmlspecialchars, qui convertit les caractères spéciaux en entités HTML, inoffensives.

Template en PHP :

```php
<?php echo $var ?>
<?php echo htmlspecialchars($var, ENT_QUOTES, 'UTF-8') ?>
```

Template Twig :

```twig
{# Par défaut Twig échappe les caractères spéciaux #}
{{ var }}
```

Langage orienté template : Boucles et conditions

PHP :

```php
<?php if ($products): ?>
  <?php foreach ($products as $pdocut): ?>
  <?php if($product->isAvailable()):>
    * <?php echo $item ?>
  <?php endif;>
  <?php endforeach; ?>
<?php else: ?>
    No product has been found.
<?php endif; ?>
```

Twig :

```twig
{% for product in products | product.isAvailable %}
  * {{ product }}
{% else %}
  No product has been found.
{% endfor %}
```

Un mot sur l'échappement, sécurité côté client

Démonstration de l'échappement et des risques à oublier (souvent) de le faire.

> *Don't try to sanitize input. Escape output*

Langage Twig

Twig vient avec son propre langage et son système de balises

Principalement **trois systèmes** de balise à retenir:

- `{# ... #}` : un commentaire ;
- `{% ... %}` : executer une instruction ;
- `{{ ... }}` : écrire quelque chose sur la sortie standard.

```
{# Execute une instruction: block, extends, function #}
{% ... %}
{# Print quelque chose}
{{ ... }}
```

Pour en savoir plus consulter la documentation.

Étendre et réécrire un template existant: extends et block

Templating : héritage et surcharge, vers une approche composants

Twig permet d'écrire des templates de manière plus sécurisée et plus lisible que PHP.

Il permet surtout d'**étendre** et de **réécrire** une partie d'un template existant.

Vous pouvez **réutiliser** des templates (avec l'instruction `extends`) et les surcharger (en redéfinir des sections) via le système d'héritage et de `block`.

Templating : héritage et surcharge

Template parent/principal:

```
{# Fichier parent.html.twig #}
{# Définition d'un block: un espace réinscriptible dans le template #}
{% block main %}
Contenu de parent.html.twig
{% endblock %}
```

Template réutilisant le template parent:

```
{# Fichier child.html.twig #}
{% extends 'parent.html.twig' %}

{%block main %}
Ce contenu écrase le contenu du template parent
{% endblock %}
```

Ce système, simple et puissant, **permet de développer une approche composant des templates et de mutualiser les éléments communs**.

Templating : héritage et surcharge

Template parent/principal:

```
{# Fichier parent.html.twig #}
{# Définition d'un block: un espace réinscriptible dans le template #}
{% block main %}
Contenu de parent.html.twig
{% endblock %}
```

Template réutilisant le template parent **et le contenu du bloc main parent** (fonction `parent()`):

```
{# Fichier child.html.twig #}
{% extends 'parent.html.twig' %}

{%block main %}
{# Utiliser le contenu du bloc parent #}
{{parent()}}
Ajouter le contenu spécifique au template
{% endblock %}
```

Inclure des templates ou fragments dans d'autres templates

Inclure des templates avec `include` :

```
{# Fichier child.html.twig #}
{% extends 'parent.html.twig' %}
{%block main %}
  {{include ('publicite.html.twig')}}
{% endblock %}
```

> *Le path des fichiers `.twig` inclus est relatif au path du dossier `templates`.*

Déclarer et manipuler des variables locales au template

Déclarer et assigner des variables dans un template.

```
{% set foo = 'bar'%}
{% set bar = {'key': 'value'} %}
```

Manipuler les données avec les filtres

Twig propose tout **un mécanisme de filtres très puissant** (*built-in* et que l'on peut développer soi-même), permettant d'écrire des templates concis et efficaces pour présenter les données.

À l'instar de la philosophie des programmes Unix, les filtres utilisent le système de *pipe* (`|`), pour pouvoir se servir d'une sortie d'un filtre comme entrée d'un autre filtre.

```
{# Exemples d'utilisation de filtre: sans paramètre, avec paramètres, enchaînés via le pipe}

{{ variable | nom_du_filtre }}
{{ variable | nom_du_filtre('param1','param2') }}
{{ variable | filtre1 | filtre2 | filtre3 }}
```

Exemple d'usage de filtres

Exercice : Générer une page web qui affiche 1000 nombres aléatoires compris entre 0 et 100. Les présenter dans un template Twig, puis **afficher uniquement les nombres supérieurs à 50 et impairs**.

```
{# On a donné à notre template Tiwg le tableau contenant les nombres aléatoires sous la variable numbers #}

<h2>Random Numbers !</h2>

{% for number in numbers %}
        {{ number }}
{% endfor %}

<h2>Nombres impairs supérieurs à 50</h2>

{% for number in numbers | filter (number => number > 50 && is odd) %}
        {{ number }}
{% endfor %}
```

Fonctions

Twig propose un ensemble de fonctions.

- `path()` : retourne une URL relative
- `url():` retourne une url absolue (avec le nom de domaine)
- `form():` permet de présenter un formulaire HTML
- `dump():` print pour le debug
- `asset(path)` : permet de charger un asset (fichier CSS, JS, image, font, etc.) en fournissant son chemin
- etc.

Twig dans Symfony

Dans une application Symfony, les templates Twig sont placés par défaut dans le dossier `/templates`. La configuration de Twig est définie dans le fichier `config/packages/tiwg.yaml`

```
twig:
    default_path: '%kernel.project_dir%/templates'

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

> *En savoir plus*

Twig dans Symfony

Dans Symfony, le Contrôleur appelle le template Twig (ici `random.html.twig`) afin de générer la réponse HTML à retourner au client. Pour cela, il utilise la méthode `render` de la classe `AbstractController`, dont il a hérité.

On en profitera pour passer les données (ici `$numbers`) que le template Twig pourra manipuler et présenter.

```php
#[Route('/random-numbers/{n}', condition: "params['n'] <= 1000", name: 'app_test_templates_random_numbers')]
    public function randomNumber(int $n)
    {
        $numbers = array_map(function () {
            return rand(0, 100);
        }, array_fill(0, $n, null));

        //On indique le chemin du template, relatif au dossier `/templates` par défaut
        return $this->render('random.html.twig', array(
            'numbers' => $numbers
        ));
    }
```

Tooling: Débug, Valider les templates

```
php bin/console debug:twig
php bin/console lint:twig templates
```

Organiser ses templates : quelques conventions, recommandations

- Nom des templates en `snake_case`, par exemple `navbar_offre_speciale.html.twig`;
- On place les templates réutilisés (`extends`) a la racine du dossier `templates`;
- On place les fragments/composants (`include`) dans un dossier `template/parts`;
- Templates spécifiques à un contrôleur:
  `template/nom_controleur/nom_du_template.html.twig`;

Outils utiles ?

Pour VS Code:

- Ajouter **auto-complétion des balises HTML** dans les fichiers `.twig` : *File* > *Preferences* > *Settings* > *Emmet* > *Include Languages* : ajouter **Item**: `twig` **Value**: `html`;
- Extension Twig Language 2, TWIG pack : syntaxe highlighting, formatting, snippets, etc.

Remarque

Avec un composant gérant le routing (symfony/routing, nikic/fast-route, slim) et un moteur de templates comme Twig, vous avez déjà des outils très puissants pour construire des sites et services web.

Adaptez également vos outils à vos besoins réels.

Ressources utiles

- Site officiel du moteur de templates Twig
- Templating Engines in PHP, article de Fabien Potencier;
- Creating and Using Templates, documentation officielle de Symfony;
- Don’t try to sanitize input. Escape output., lecture recommandée sur les bonnes stratégies à employer pour sécuriser un contexte web;
- htmlspecialchars, fonction native de PHP. **A utiliser pour échapper les caractères spéciaux** dans un template de page HTML;
- Front-end Tools: Handling CSS & JavaScript

Exercices

Réaliser les exercices fournis avec le module 8.

TP Fil Rouge - Système de billetterie

Poursuivre le TP fil rouge (Partie 3), correspondant au module couvert.