Secteur Tertiaire Informatique
Filière « Etude et développement »

Séquence « Développer des pages Web »

# TP PHP

Apprentissage

**Mise en pratique**

Evaluation

# TABLE DES MATIERES

# 1. ATELIER TWIG

**Twig** est un générateur de Template en **PHP**, il peut être utilisé avec un Framework (c'est le générateur de Template attitré de **Symfony**) mais aussi dans un projet web en PHP indépendant de tout Framework.

Ce TP sera l'occasion de découvrir l'installation de Twig dans un projet web PHP sans avoir recours à un Framework.

Pour commencer, vous devez installer **Composer** sur votre ordinateur. **Composer** est un **gestionnaire de dépendances** pour les projets PHP. C'est l'alter-ego de **Npm** pour les projets Front qui reposent sur **Node.js**.

Si vous travaillez avec un OS **Windows**, il existe un installateur qui vous permettra d'installer Composer **globalement** sur votre machine. ( RDV sur ***https://getcomposer.org/***)

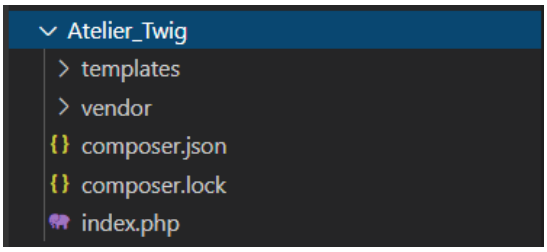
Lorsque Composer sera installé sur votre ordinateur. Vous devrez créer par exemple un répertoire de projet sur votre serveur web : **Atelier_Twig**

En suite, vous devrez ouvrir une fenêtre **Terminale** au niveau de ce dossier et vous exécuterez la ligne de commande :

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

Ce qui aura pour effet d'installer Twig pour votre projet. A la fin du processus d'installation, un dossier **vendor** aura été créé ainsi qu'un fichier **composer.json** et aussi un fichier **composer.lock** qui permet de verrouiller toutes les versions des dépendances installées par notre projet.

Vous allez devoir créer un script : **index.php** ainsi qu'un dossier **templates** :




Le script *index.php* devra contenir le code suivant :

```php
require_once 'vendor/autoload.php';

$loader = new \Twig\Loader\FilesystemLoader('templates');
$twig = new \Twig\Environment($loader, [
    'cache' => false,
]);

$prenom = "Sacha";
$nom = "restoueix";
$fonction = "Formateur DWWM";

echo $twig->render('home.html.twig', [
    'prenom' => $prenom,
    'nom'    => $nom,
    'fonction' => $fonction
]);

?>
```

Dans le dossier **templates** vous allez créer le fichier *home.html.twig* qui comportera le code suivant :

```twig
<h1>Bienvenue {{prenom}} {{nom | upper}}, vous êtes {{fonction}} </h1>

Date du jour : {{ "now"|date('d/m/Y') }}
```

Lorsque vous chargerez le script *index.php* dans votre Navigateur vous obtiendrez ce rendu :

**Bienvenue Sacha RESTOUEIX, vous êtes Formateur DWWM**

Date du jour : 31/01/2021

Voilà, vous avez réalisé votre première *mini application* qui repose sur le générateur de vue **Twig**.

Twig offre un véritable langage de programmation avec des **boucles**, des **conditions**, des **filtres** …

Il permet d'alléger le code de façon significative par rapport au code PHP qui est très verbeux. Au final c'est bien du code PHP qui est généré mais il est optimisé.

De façon native, **Twig** offre une protection des données en échappant par défaut toutes les valeurs des variables qui comportent des caractères spéciaux. Ce qui permet de lutter efficacement contre les **injections XSS**.

Si vous ne voulez pas que Twig échappe vos variables qui contiennent du code HTML, il faut lui préciser en utilisant le filtre **raw** :

```
{{ ma_variable_html|raw }}
```

Si vous voulez utiliser un commentaire dans votre script.twig vous devez utiliser la syntaxe suivante :

```
{# Ceci est un commentaire Twig #}
```

Lorsque vous voulez **déclarer des variables** et les **concaténer** dans **Twig** :

```
{% set message = 'Salut' %}
{% set nom = 'Sacha' %}

{{ message ~ nom|lower }}    {# Salut sacha #}
```

## Les Structures de contrôle avec Twig :

Pour utiliser une structure de contrôle **If …elseif … else … endif** avec Twig :

Si vous avez déclaré au préalable une variable **$membre** comme il suit au niveau de votre contrôleur :

```
$membre = ["age" => 52];
```

Du côté Template vous pourrez utiliser la **structure de contrôle** suivante en utilisant la syntaxe **Twig** :

```
{% 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 %}
```

Pour utiliser la boucle **For** avec la syntaxe **Twig** :

Dans votre contrôleur définissez la liste :

```
$liste = ["Tigre", "Lion", "Panthère", "Ocelot", "Jaguar"];

echo $twig->render('home.html.twig', [
    'felins'     => $liste
]);
```

Dans votre Template utilisez la syntaxe Twig pour itérer sur cette collection :

```
<ul>
  {% for felin in felins %}
    <li>{{ felin }}</li>
  {% else %}
    <li>Pas de félin trouvé.</li>
  {% endfor %}
</ul>
```

La boucle **For** définit une variable **loop** qui possède différents **attributs** qui peuvent être très utiles :

```
{{ loop.index }}
```

Le numéro de l'itération courante (en commençant par 1).

```
{{ loop.length }}
```

Le nombre total d'itérations dans la boucle.

**...**

Ainsi en utilisant notre tableau de félins on peut définir un rendu côté Vue en utilisant le test suivant :

```
{% for felin in felins %}
  <span class="{% if loop.index is even %}pair{% else %}impair{% endif %}">
    {{ felin }}
  </span>
{% endfor %}
```

On peut aussi utiliser le test

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

Qui permet d'alléger la syntaxe PHP équivalente :

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

**<u>La Notion d'Héritage avec Twig :</u>**

Parlons maintenant de **l'héritage** à la mode **Twig** :

Dans beaucoup de langages nous avons la possibilité **d'inclure** du code afin d'éviter sa répétition. Notre projet peu à tout moment devenir un projet conséquent avec le temps et si les bonnes pratiques ne sont pas mises en œuvre dès le départ il sera extrêmement difficile de faire demi-tour.

Si on doit changer une valeur à un endroit dans notre code qui est recopié partout, nous allons devoir la changer autant de fois qu'il y a de copié/collé.

Il faut avouer que c'est plutôt dommage quand on sait qu'avec **Twig** on peut facilement centraliser notre code en un fichier puis **inclure** ce fichier chaque fois que bon nous semble.

**L'héritage** avec **Twig** repose sur le même concept que celui du développement classique, il n'y a rien de bien nouveau : nous avons établi une charte graphique pour notre site et nous voulons l'utiliser partout dans notre application.

À la façon du développement, dans lequel nous avons une **classe** qui sera étendue par sa/ses filles, avec **Twig** nous allons avoir un Template **père** dans lequel il y aura des "**blocks**" à remplir et il sera étendu et rempli par son/ses **fils** pour former une page complète.

La différence réside dans le fait que là où *l'include* en PHP ne permet que la modification à l'endroit où il est écrit, le fils d'un Template Twig peut remplir plusieurs "blocks" (plusieurs trous).

Le Template **père** s'appelle en général le "*layout*". On aura par exemple le fichier *layout.html.twig* :

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>{% block title %}Notre site{% endblock %}</title>
  </head>

  <body>
    {% block body %}
    {% endblock %}
  </body>
</html>
```

Voici un exemple de Template **fils** nommé accueil qui comblerait le block body :

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

{% block title %}{{ parent() }} - Accueil{% endblock %}

{% block body %}
  Voici le body de notre page.
{% endblock %}
```

Dans le **fils** nous pouvons voir le mot "**extends**" qui permet de faire référence au Template **père**.

Nous pouvons aussi voir le "**block title**" et le "**block body**" dans le Template **père** que nous retrouvons dans le **fils**.

Comme expliqué, les blocks dans le **fils** servent à remplir le Template **père**, le block title **fils** prend alors la place du block title dans le Template **père**.

Nous pouvons donc en déduire que le nom d'un block se définit par {% block son_nom %} et il est délimité par ce même block et {% endblock %}.

Dans le **fils** nous retrouvons cette même notation avec un {% block son_nom %} et {% endblock %}.

Twig offre aussi la possibilité de mettre en **cache** les modèles compilés sur le système. Ainsi, la compilation est faite une seule fois et cela améliore la performance de notre site.

Pour cela il vous suffit de créer un répertoire **repCache** à la racine de votre projet et d'indiquer à Twig que vous voulez activer la mise en cache sur votre projet.

Pour activer le système de cache de Twig, nous devons renseigner le nom du répertoire de cache dans notre fichier « *index.php* ».

```
…
$twig = new Twig_Environment($loader, [
    'cache' => 'repCache',
]);
…
```

Vous verrez qu'après avoir actualisé votre **Navigateur**, un dossier est créé automatiquement dans le dossier de mise cache **repCache** et que dans ce dossier nous avons un fichier « *php* » qui contient le contenu de notre page.

Afin de bien prendre en main toutes ces notions, je vous propose de réaliser ce petit exercice.

Pour que vos modifications soient systématiquement prises en compte pendant la phase de développement, je vous conseille, avant de démarrer, de supprimer la mise en cache offerte par **Twig** en revenant à :

```
'cache' => false,
```

Vous allez devoir implémenter un petit site web qui présentera les différents langages du Web, sans être exhaustif.

Une barre de navigation qui sera commune à toutes les vues offrira une navigation aisée sur le site. Les différentes rubriques [**HTML**, **CSS**, **JavaScript**, **JQuery**] seront

contenues dans un **tableau** que vous devrez parcourir pour créer cette barre de navigation.

Vous devrez jouer sur **l'héritage de Template** qu'offre **Twig** pour implémenter cette partie de la solution.

Vous devrez intégrer à votre solution une feuille de style générale, vous intégrerez aussi **BootStrap** ou un thème BootStrap de votre choix, enfin vous devrez intégrer la Bibliothèque **JQuery** à cette application.

Le rendu général aura cette forme :


Chaque section devra présenter le langage qui lui est affecté.

Ce TP vous a permis de vous familiariser avec la syntaxe de **Twig** et avec **Composer** ce qui sera un acquis important avant d'aborder l'étude du célèbre Framework PHP **Symfony**.

**CREDITS**

**ŒUVRE COLLECTIVE DE l'AFPA**
**Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services**

**Equipe de conception (IF, formateur, mediatiseur)**
Formateur : Sacha RESTOUEIX

**Date de mise à jour** : 28/01/2021