Prise en main du moteur de Template Twig

L'objectif

Ce pas à pas apprend à utiliser le moteur de *template* **Twig**, qui permet de séparer l'affichage, c'est-à-dire la partie qui génère le code HTML présenté à l'utilisateur, du code métier. L'objectif est de créer une page web présentant les livres d'une bibliothèque depuis **PHP**, en utilisant des *templates* Twig.

Les outils

- un navigateur web
- un IDE (PHPSTORM par exemple)
- un serveur web **Apache** avec PHP

Phase 1 - Installation

Twig est le moteur de *template* intégré dans le *Framework* **Symfony**, mais il peut être utilisé indépendamment de Symfony comme c'est le cas dans ce « How to ». La documentation de Twig est disponible à l'URL suivante : https://twig.symfony.com/.

Pour exécuter le moteur de *template*, il faut un serveur web Apache avec PHP qui ne doit pas être antérieur à la version 5.2.7.

Si le gestionnaire de paquets **Composer** est déjà installé, se placer dans le répertoire de l'application web (dans cet exemple, c:\xampp\htdocs\appli), puis exécuter la commande composer pour télécharger la dernière version de Twig :

cd c:\xamp\htdocs\appli

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

Il est également possible de créé un fichier composer.json contenant les dépendances :

```
{
  "require": {
    "twig/twig": "v3.7.1"
  }
}
```

Puis de lancer la commande :

```
composer install
```

Le répertoire du projet web contient à présent un répertoire vendor comportant la bibliothèque Twig ainsi qu'un script autoload.php. La version de Twig testée dans ce document est la 3.7.1 août 2023.

Attention aux droits : Twig utilise un répertoire cache pour stocker les fichiers PHP générés à partir des *templates* et donc le répertoire c:\xampp\htdocs\appli\cache doit être accessible en écriture pour le serveur web.

Phase 2 - Création d'un Template

Le *template* est utilisé pour générer l'interface utilisateur, car séparer la présentation des traitements facilite le développement du projet et sa maintenance. La page web reçue par le navigateur est générée côté serveur à partir de deux fichiers :

- Le *template* ou gabarit (vue) est un fichier avec l'extension twig, qui contient des parties statiques (code HTML) ainsi que des parties dynamiques permettant de sélectionner et intégrer des données dans la page HTML. Il utilise un langage simple et concis pour parcourir les données et extraire l'information à afficher (boucles, conditions, variables, fonctions, filtres).
- Un script PHP (contrôleur) qui définit les données qui seront passées au *template*.

Le script PHP appelé par le navigateur, récupère les données et les met à disposition du *template* (figure 1). Le *template* est analysé et compilé par le moteur de *template* en un fichier PHP. Ce dernier est placé dans le répertoire de cache afin de limiter la consommation de ressources sur le serveur. Une fois ce fichier exécuté, le code HTML est généré et transmis au navigateur.

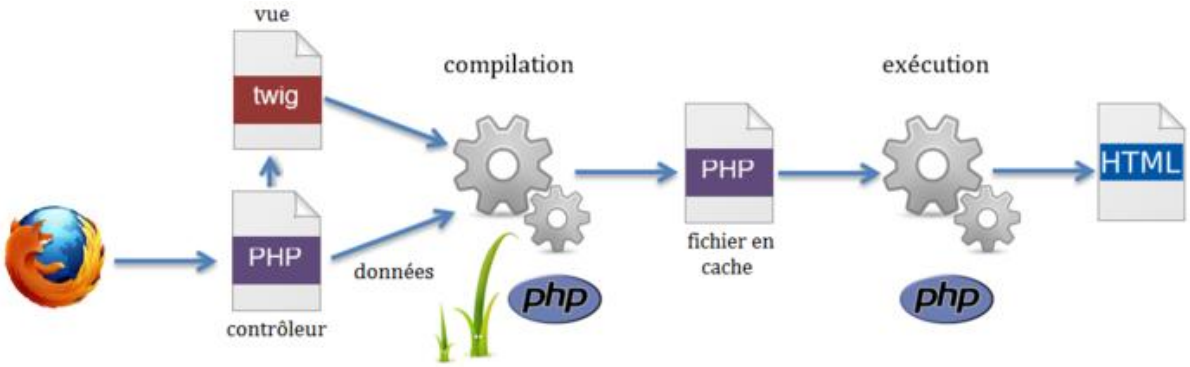

Figure 1: Principe de fonctionnement de Twig.

Nous allons créer un fichier biblio.twig qui affichera les données transmises par le script biblio.php, qui sera écrit dans la phase suivante. Placer le fichier biblio.twig dans le répertoire vue de l'application web. Il comportera du code HTML ainsi que des parties dynamiques marquées par deux types de délimiteurs {{ exp }} et {# ... #}. Le premier affiche le résultat de l'évaluation de l'expression exp, ce qui permettra d'afficher les données de manière dynamique. Le second ajoute une ligne de commentaire.

Délimiteur	Description	Exemple	Équivalent PHP
{{ exp }}	Afficher le résultat de l'évaluation de l'expression exp	{{ x }}	<?php echo $x ?> ou <?= $x ?>
{# ... #}	Commentaire twig (supprimé lors de la compilation)	{# commentaire #}	/* commentaire */
{% ... %}	Exécuter l'instruction (if, for, ...)	{% if titre|length > 25 %}	

Le fichier biblio.twig (figure 2) contient un code HTML statique, ainsi que du code twig surligné en rouge. Les parties dynamiques du code entre délimiteur double accolades seront remplacées lors de l'exécution par les valeurs des variables titre_doc, titre_page et date. Il est possible d'écrire des commentaires dans les *templates* en les plaçant entre accolades et dièses. Le commentaire dans ce fichier indique l'emplacement où les titres et noms de livres seront mis en page dans les phases suivantes.

Figure 2: template biblio.twig

Phase 3 - Création du code PHP

Le fichier biblio.php, placé à la racine de l'application web, doit récupérer les données, initialiser Twig, stocker la configuration, charger et compiler le *template* biblio.twig et retourner le résultat au navigateur.

Pour simplifier l'exemple, la fonction getData()du script data.php, retourne des données statiques plutôt que d'interroger une base de données. Ce fichier, placé dans le répertoire modele, est inclus dans biblio.php. Twig travaille avec des tableaux de données. Dans le *template* Twig {{ titre_doc }} fait référence à la valeur de la clé titre_doc du tableau associatif des données.

```
<?php
// data.php
function getData(){
    // tableau associatif des données
    $data = array(
        'titre_doc' => 'Bibliotheque',
        'titre_page' => 'Liste des livres',
        'date'  => date("d/m/Y"),
        // pour simplifier l'exemple, les données sont définies
        // statiquement (généralement elles sont extraites d'une BD)
        'biblio' => array(
            array('titre'=>'N ou M', 'nom'=>'Christie', 'prenom'=>'Agatha'),
            array('titre'=>'1984', 'nom'=>'Orwell', 'prenom'=>'George'),
            array('titre'=>'Dune', 'nom'=> 'Herbert', 'prenom'=>'Frank')
        )
    );
    return $data;
}
```

Une fois les données récupérées et stockées dans la variable $donnees dans biblio.php, le script inclut l'autoloader, puis définit le mode de chargement du *template* et son emplacement, en créant une instance de la classe Twig_Loader_Filesystem. Le *template* sera chargé depuis le chemin passé en paramètre, c'est-à-dire le répertoire vue placé au même niveau de l'arborescence que biblio.php. Nous avons défini deux tableaux d'options pour les phases de développement et de production. Pendant le développement, le système de cache est désactivé en attribuant la valeur false à l'option cache. Lorsque l'application sera en production, Twig stockera dans le répertoire cache les fichiers PHP générés. L'option autoescape permet de protéger automatiquement les sorties contre les attaques de type XSS. D'autres options sont disponibles, elles sont décrites dans le tableau d'options ci-après.

Option	Valeur par défaut	Description
charset	utf-8	Jeu de caractères des *templates*
cache	false	Chemin du répertoire de cache
strict_variables	false	false = ignorer les variables ou fonctions inexistantes (remplacé par null) ; true = lancer une exception
autoescape	true	true = protection automatique des sorties (la stratégie de protection peut être définie : css, url, htm_attr) ; false = pas de protection automatique

Le script biblio.php crée une instance de la classe Twig\Environment, puis il charge et compile le *template* biblio.twig avec les données du tableau $donnees. Enfin, l'instruction echo retourne la page HTML au navigateur.

```
<?php

// biblio.php
/* récupérer le tableau des données */
require 'modele/data.php';
$donnees = getData();

 /* inclure l'autoloader */
require_once 'vendor/autoload.php';
/* templates chargés à partir du système de fichiers (répertoire vue) */
$loader = new Twig\Loader\FilesystemLoader('vue');
/* options : prod = cache dans le répertoire cache, dev = pas de cache */
$options_prod = array('cache' => 'cache', 'autoescape' => true);
$options_dev = array('cache' => false, 'autoescape' => true);
/* stocker la configuration */
$twig = new Twig\Environment($loader);
/* charger+compiler le template, exécuter, envoyer le résultat au navigateur */
echo $twig->render('biblio.twig', $donnees);
```

Phase 4 - Exécution

Pour tester l'exemple, il suffit d'utiliser l'URL suivante dans le navigateur : http://localhost/appli/biblio.php. Lorsque le navigateur demande le script biblio.php, les données et la vue biblio.twig sont compilées dans un fichier PHP (figure 3) ; c'est ce fichier qui est exécuté et qui génère le code HTML présentant les données.

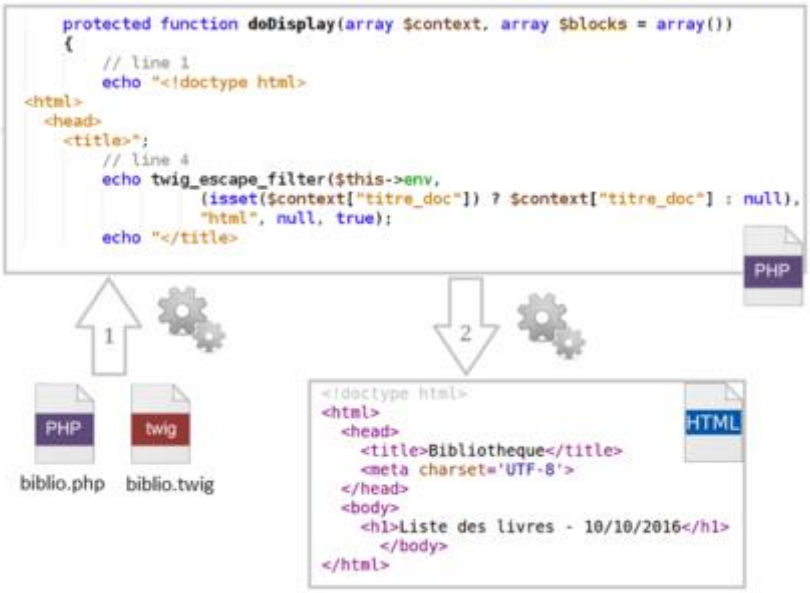

Figure 3: Processus d'exécution

Le code dynamique du *template* a été remplacé par le mot Bibliotheque dans le titre du document (onglet dans la figure 4). Le titre de niveau un (h1) contient un titre suivi de la date d'exécution du script. Le résultat HTML ne comporte pas encore de liste de livres, car elle sera ajoutée dans une phase ultérieure. Si vous avez fait une erreur de syntaxe dans le *template*, vous obtiendrez une erreur 500 lors de l'exécution, vérifiez les logs d'erreur du serveur Apache pour déterminer la cause du problème.

Figure 4: Résultat de l'exécution de biblio.php

Phase 5 : Ajout d'une condition

Nous allons modifier le *template* biblio.twig, pour afficher un message si la variable biblio ne contient pas de données. Pour cela, il faut utiliser les structures conditionnelles if et else entre les délimiteurs d'exécution d'instruction {% ... %}.

```
<!doctype html>
<html>
  <head>
    <title>{{ titre_doc }}</title>
    <meta charset='UTF-8'>
  </head>
  <body>
    <h1>{{ titre_page }} - {{ date }}</h1>
    {% if biblio is not empty %}
 {# afficher les titres et noms des livres #}
    {% else %}
        Aucun livre dans la bibliothèque.
    {% endif %}
  </body>
</html>
```

Les expressions dans les conditions utilisent des opérateurs et des tests définis dans les tableaux ci-après. L'expression du listing combine l'opérateur is not et le test empty. Si la variable biblio n'est pas vide, les titres et noms de livres seront affichés dans la phase suivante, sinon un message indique qu'il n'y a pas de livre dans la bibliothèque (figure 5).

Figure 5: Résultat lorsqu'il n'y a pas de données.

Opérateurs	Types d'opérateurs
+ - * / %	Mathématiques
and or not	Logiques
== != >= <= > <	Comparaison
in	Booléen (ex : a in b est vrai si a est dans b)
is is not	Test
..	Création de séquence
|	Filtre
~	Concaténation
? :	Ternaire

Tests	Description
defined	Vrai si la variable est définie dans le contexte courant
empty	Vrai si la variable est vide (null, false, '', tableau vide)
null	Vrai si la variable est à null
even / odd	Vrai si la variable est paire/impaire
iterable	Vrai s'il est possible d'itérer sur la variable (tableau, objet traversable)

Phase 6 - Parcours d'un tableau

Dans cette phase, nous allons parcourir les données du tableau biblio, afin d'afficher les titres et les noms d'auteurs. La boucle for parcourt chaque item du tableau biblio. L'item étant lui-même un tableau, pour accéder au nom de l'auteur du livre, il faudra utiliser le nom du tableau suivi du caractère point et de la clé, c'est-à-dire item.nom (figure 6). La figure 7 montre le résultat dans le navigateur. Les filtres utilisés dans le listing et leurs effets sont présentés dans la dernière phase.

```
<!doctype html>
<html>
  <head>
    <title>{{ titre_doc }}</title>
    <meta charset='UTF-8'>
  </head>
  <body>
    <h1>{{ titre_page }} - {{ date }}</h1>
    {% if biblio is not empty %}
    <table>
        <thead>
            <tr><th>Titre</th><th>Auteur</th></tr>
        </thead>
        <tbody>
        {# afficher les titres et noms des livres #}
        {% for item in biblio|sort %}
```

```
            <tr>
                <td>{{ item.titre }}</td>
                <td>{{ item.nom|capitalize }}</td>
            </tr>
        {% endfor %}
        </tbody>
    </table>
    {% else  %}
        Aucun livre dans la bibliothèque.
    {% endif %}
  </body>
</html>
```

Figure 6: Parcours du tableau.

Phase 7 - Ajout de filtres

Twig propose une trentaine de filtres qui peuvent être appliqués à une variable ou à une section de code. Il est possible de chaîner les filtres. Une partie des filtres est listée dans le tableau ci-après. Dans le listing, nous utilisons capitalize pour mettre en majuscule la première lettre du nom de l'auteur, ainsi que le filtre sort pour trier le tableau de données avant son affichage.

Filtre	Type	Rôle du filtre
number_format	Nombre	Retourne le nombre formaté, argument 1 = nb de décimales, argument 2 = séparateur décimal ex : {{ nb|number_format(2, ',') }} format avec 2 chiffres après la virgule
round	Nombre	Retourne le nombre arrondi, argument 1 = précision (défaut), argument 2 = type d'arrondi (ceil, floor ou common qui choisit l'arrondi supérieur ou inférieur en fonction de la valeur) ex : {{ nb|round }} ou {{ nb|round(2,'ceil') }}
capitalize	Chaîne	Retourne la chaîne avec le premier caractère en majuscule, les autres en minuscules ex : {{ titre|capitalize}
lower	Chaîne	Retourne la chaîne en minuscules
upper	Chaîne	Retourne la chaîne en majuscules
length	Chaîne, tableau	Retourne la taille de la chaîne ou du tableau
keys	Tableau	Retourne les clés d'un tableau
join	Tableau	Retourne la concaténation des valeurs des cases d'un tableau, argument 1 = délimiteur ex : nom_tab|join ou nom_tab|join(',')
sort	Tableau	Tri de tableau. ex : {% for item in tab|sort %} ... {% endfor %}
date	Date	Formate la date. ex : {{ livre.parution|date("d/m/Y") }}

Le résultat

La figure 7 montre le résultat obtenu une fois toutes les phases réalisées. La page web affiche la liste des livres définie dans le tableau de données (fichier data.php). Les variables du *template* biblio.twig ont été remplacées dans les éléments title et h1. Les lignes du tableau HTML ont été générées dynamiquement par la boucle for. La liste des livres n'est pas affichée dans l'ordre de déclaration du tableau de données, car un filtre de tri a été appliqué dans le *template*. Le filtre capitalize a permis d'uniformiser l'affichage des noms en changeant le premier caractère en majuscule, c'est ce qui explique la majuscule à Orwell qui n'en avait pas dans le tableau de données. Si le fichier data.php ne contient pas de clé biblio ou que cette clé contient un tableau vide, alors un message est affiché (figure 5).

Figure 7: Résultat présentant les données.