esprit
Se former autrement

Le moteur de templates : Twig

UP Web

AU: 2020/2021

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
CONFÉRENCE DES GRANDES ÉCOLES
EXCELLENCE FOR A COMPLEX WORLD
European Accreditation of Engineering Programmes EUR-ACE®
délivrée par CTI Commission des Titres d'Ingénieur
cdio
CONCEIVE DESIGN IMPLEMENT OPERATE

Plan

1. Qu'est ce qu'un moteur de templates
2. Le moteur de templates Twig
3. Syntaxe de base
    a. Déclaration et affichage des variables
    b. La concaténation
    c. La structure conditionnelle
    d. La structure itérative
    e. Les filtres
    f. Les fonctions
    g. Les variables globales
    h. Les liens
4. Ajout de fichiers

1. Qu'est ce qu'un moteur de templates

- PHP peut être considéré comme un moteur de template
- Il est possible de mélanger du PHP avec du code HTML mais il reste très verbeux et peu pratique pour certaines tâches.

Exemple:

```
<h1>Bienvenue</h1>
<p>Bienvenue sur mon site <?= isset($person['name']) ?
 htmlentities($person['name']) : '' ?></p>
<?= markdown($person['bio']) %>
<?php
$content = ob_get_clean();
require 'layout.php';
?>
```

- Ce code est difficilement lisible

=> **séparer la présentation des traitements facilite le développement du projet et sa maintenance.**

1. Qu'est ce qu'un moteur de templates

Pourquoi utiliser un moteur de templates ?

- Séparer le traitement de l'affichage
- Permettre aux designers de développer rapidement des gabarits sans spécialement connaître le langage utilisé
- Minimiser le code et le rendre plus clair

Les principaux moteur de templates php

- php
- raintpl
- smarty
- twig
- mustache
- savant3
- talus..

Inconvénient :

- Un peu plus lent à exécuter (cache obligatoire)

2. Le moteur de template Twig

- Twig est un moteur de templates développé par SensioLabs,apparu en 2009
- Utilisé par Symfony et d'autre projet PHP
- La Version actuelle: Twig 3
- Un projet symfony 4 offre la version 3 et la version 2.x de Twig ,la version de Twig utilisée dépendra de la version de PHP

Twig 2.x	Twig 3
• PHP>=7.1.3 et <=7.2.4 • Des fonctionnalités obsolètes sont supprimées dans la version 3 (voir la documentation officielle https://twig.symfony.com/doc/2.x/deprecated.html)	• PHP >=7.2.5

2. Le moteur de template Twig

Pour travailler avec une version spécifique de Twig dans un projet Symfony, il suffit de:

- Modifier le fichier composer.json (choisir la version 2.x ou 3 de Twig)

  "twig/twig": **"^2.12|^3.0"**

- Mettre à jour les dépendances :composer update

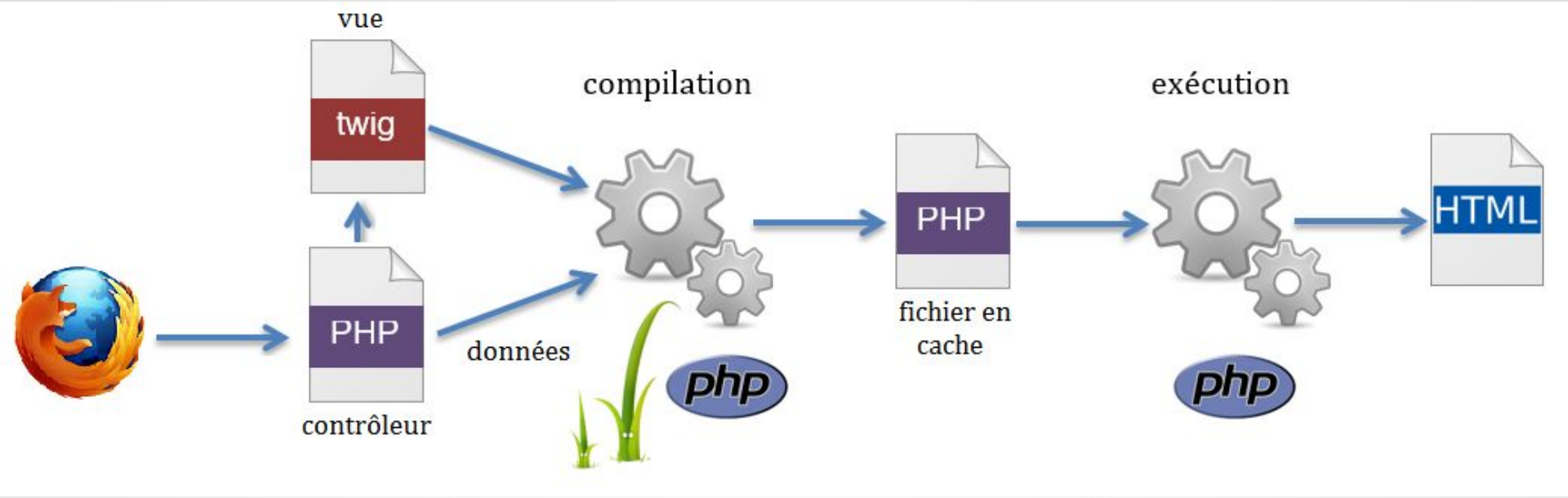
2. Le moteur de template Twig

Les avantages de Twig:

- Permet de séparer la présentation des données du traitement.
- Permet la personnalisation de la page web.
- Permet de rendre les pages web plus lisibles, plus claires.
- Apporte de nouvelles fonctionnalités (comme l'héritage des templates, les filtres…).
- Apporte plus de sécurité.

2. Le moteur de template Twig

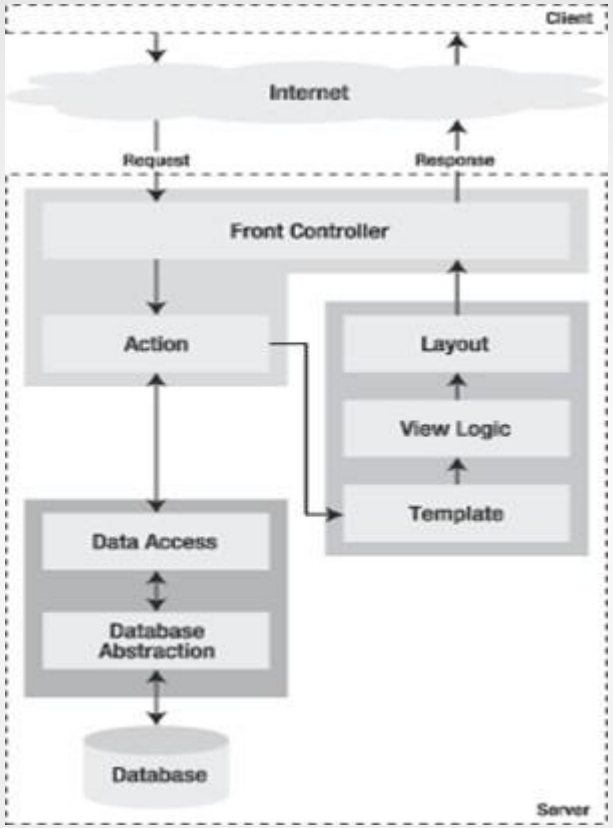
Fonctionnement de twig



Principe de fonctionnement de Twig.

2. Le moteur de template Twig

Retourner un Template Twig



2. Le moteur de template Twig

Retourner un Template Twig

- Depuis le contrôleur, on utilise la méthode **render()** pour retourner une interface

- la méthode render() prend en paramètre:
  1. Le chemin vers le template:
     /Template/vue
  2. Un tableau des paramètres à afficher dans le twig

```
//Dans le controleur
return $this->render( view: 'another/index.html.twig', [
    'controller_name' => 'AnotherController',
]);
```

le répertoire contenant les templates propre à ce contrôleur

Nom de la vue

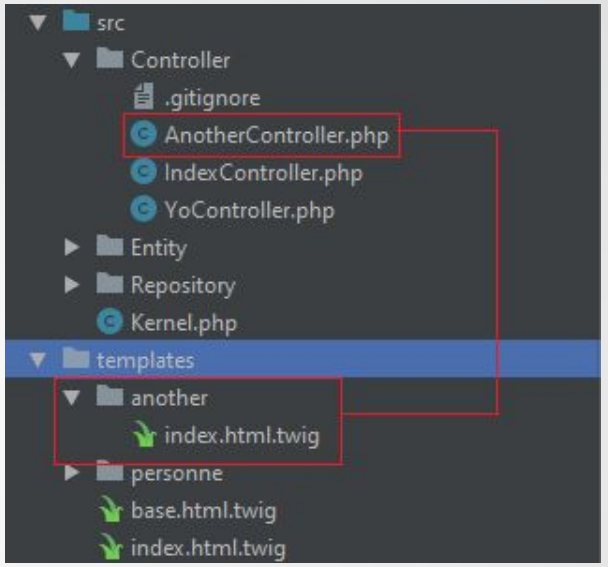
Tableau des Paramètres envoyé du Controller au vue

2. Le moteur de template Twig

Emplacement des templates

- Les vues sont placées sous le dossier Template
- Il est recommandé de créer un répertoire sous dossier template pour chaque contrôleur

- On utilise les deux extensions .html et .twig



3. La Syntaxe de base

Trois types de syntaxes :
- Syntaxe de base pour afficher des variables

- {{ … }} affiche quelque chose

- Syntaxe de base pour les structures de contrôle et les expressions

- {% … %} exécute une action

- Syntaxe pour les commentaires

- {# … #} définit un commentaire

3. La Syntaxe de base

a. Affichage et déclaration des variables

- Afficher une variable se fait avec les doubles accolades « {{ … }}»
- Affichage d'une variable simple: **{{ title}}** est esquivant à <?php echo $title ?>
- Affichage d'index d'un tableau **{{ T['i']}}** est équivalant à <?php echo T['i'] ?>
- Affichage l'attribut d'un objet, dont le getter respecte la convention $objet->getAttribut()

twig : Identifiant : {{ user.id }} => php: **Identifiant : <?php echo $user->getId(); ?>**

3. La Syntaxe de base

Déclaration d'une variable

- une variable
  {% set pi = 3.14 %} {% set foo = 'foo' %}
- tableau simple avec une série de valeurs:
  {% set tableau=[1, 2, 3] %}
- tableau avec des clés:
  {% set tableau={key1:value1, key2:value2} %}
- tableau avec valeur et clé:
  {% set foo = [3, {"mot": "soleil"}] %}
  afficher le contenu de 'mot' donc 'soleil':
  {{ foo[1]['mot'] }}
- déclarer 2 variables en même temps // foo='foo' // bar='bar':
  {% set foo, bar = 'foo', 'bar' %}
- foo contient le texte entre les 2 balises:
  {% set foo %}
  ** **
  **… **
  {% endset %}

3. La Syntaxe de base

b. Concaténation

- Concaténer deux variables dans le Template twig:

{{ "Description du produit:" ~ produit.description }}

{{ var1 ~ var2 }}

3. La Syntaxe de base

c. La structure conditionnelle:{% if … %} … {% endif %}

- **Une condition avec empty**

```
{% if produits is empty %}
  il n'y a plus de produit
{% endif %}
```

- **Une condition avec and, or ,defined**

```
{% if ((a==1 and b>0) or not c==0) and d is defined %}
      {% set resultat = (d + a * b) / c %}
  {{ resultat }}
{% endif %}
```

- **Une condition avec start , ends**

```
{% if 'Fabien' starts with 'F' %}
  commence par F
{% endif %}
```

```
{% if 'Fabien' ends with 'n' %}
      Finis par n
{% endif %}
```

3. La Syntaxe de base

- Une condition avec **matches:** permet dede determiner si une variable respecte un motif donne par une expression régulière

```
{% if phone matches '/^[\\d\\.]+$/' %}
        format telephone ok
{% endif %}
```

- Une condition avec **not in**

```
{% if 5 not in [1, 2, 3] %}
        5 non présent
{% endif %}
```

- **Une condition avec else if**

```
{% if var  is odd %}
  yes
{%else%}
no
   {% endif %}
```

ou

```
{{ var is odd? 'yes': 'no' }}
```

3. La Syntaxe de base

- **Tests utiles:**

  is null: si est null ;

  is constant: comparer si est une constante ;

  is divisible by(*x*): si est divisible par *x* ;

  is even: si est pair ;

  is odd: si est impair ;

  is iterable: si est du type itérable (comme une liste) ;

  is same as: comparer 2 variables (en php correspond ===).

- La liste des test utiles: https://twig.symfony.com/doc/3.x/tests/index.html

3. La Syntaxe de base

d. La structure itérative: for

- Parcourir un tableau associatif

```
{% for produit in produits %}
    {{ produit.nom }}
{% endfor %}
```

- Parcourir un tableau indexé: (on peut utiliser l'operateur ´ **..** pour definir un intervalle)

```
{% for i in 0..9 %}    // pareil que {% for i in range(0, 9) %}
    {{ i }}
{% endfor %}
```

// affiche 0123456789

3. La Syntaxe de base

d. La structure itérative: for

- Parcourir un tableau associatif avec une condition:

```
{% for produit in produits if produit.etat==1% }
        {{ produit.nom }}
{% endfor %}
```

- Parcourir un tableau associatif avec une condition vide:

```
{% for article in articles %}
        {{ article.nom }}
{% else %} pas d'article trouvé
{% endfor %}
```

3. La Syntaxe de base

d. La structure itérative: for

- clés et valeurs:

{% for key, value in table %}

{{ key }} {{ value }}

{% endfor %}

Exemple:

```
{% for cle, valeur in tableau %}
{{ cle ~ ' : ' ~ valeur }} <br>
    {# result:
    0:3
    1:6
    2:57 #}
{% endfor %}
```

3. La Syntaxe de base

e. Les filtres

- Permettant de formater et modifier l'affichage d'une donnée´
- Pouvant prendre un ou plusieurs paramètres `
- Syntaxe : {{ variable | fonction filtre[paramètres] }}
- On peut appliquer des filtres sur une variable à afficher, sur une variable d'une condition IF ou d'une boucle FOR…
- **Appliquer un seul filtre**

```
{% for i in 0..listetableau|length %}
    {#traitement#}
{% endfor %}
```

- **Appliquer plusieurs filtres**

```
{{"les fleurs du mai"|trim|upper}}
```

- **Appliquer un filtre sur un texte long avec apply**

```
{% apply upper %}
    This text becomes uppercase
{% endapply %}
```

3. La Syntaxe de base

e. Les filtres (exemples)

Les filtres	Description
upper()	Affiche la variable en majuscules {{“hello”|upper}}=>HELLO
lower()	Affiche la variable en minuscules {{“HELLO”|lower}}=>hello
trim()	Supprime les caractères spéciaux indiqués du début et de la fin d’une chaîne de caractères {{“hello world .”|trim(‘.’)}} =>hello world
Slice(start, length)	Extrait les éléments de la position start et de nombre length {% **for** i **in** [1, 2, 3, 4, 5]|slice(1, 2) %} *{# will iterate over 2 and 3 #}* {% **endfor** %} {{ **'12345'**|slice(1, 2) }} *{# outputs 23 #}*
length()	calcule le nombre d’élèments d’un tableau ou le nombre de caractères d’une chaîne

3. La Syntaxe de base

e. Les filtres (exemples)

Les filtres	Description
Date()	Formate la date selon le format donné en argument {{post.published_at|date(“m/d/Y”)}}
date_modify()	Formate la date selon la chaine donnée en paramètre {{post.published_at|date_modify(“+1 day”)}}

- La liste complète des filtres: https://twig.symfony.com/doc/3.x/filters/index.html

3. La Syntaxe de base

f. Les fonctions

- Permettant d'effectuer un traitement

Exemples:

- **Dump()**: affiche tout le détail d'un objet ou d'un tableau

```
{{dump(entities)}}
```

- **Max():**

```
{% set tab = [1, 5, 2, 3] %}
{{ max(tab) }} =>5
```

- **Date():** permet de convertir l'argument en date afin de pouvoir comparer les dates

```
{% if date(club.created_at) < date() %}
 {# traitement #} {% endif %}
```

Liste des fonctions: https://twig.symfony.com/doc/3.x/functions/index.html

3. La Syntaxe de base

g.Les variables globales

- **app**
  Variable globale qui va nous permettre de récupérer des informations de notre application.
- **app.environment**
  Récupère l'environnement actuel pour savoir si vous êtes sur l'interface de production ou de développement.
- **app.debug**
  Permet de savoir si le mode debug est activé ou non (retourne un boolean).
- **app.user**
  Récupère les informations de l'utilisateur courant, c'est ni plus ni moins que l'entité User.

3. La Syntaxe de base

g.Les variables globales

- **app.request**
  Retourne la séquence de tous les éléments disponibles de la requête HTTP.
- **app.request.query**
  Permet d'accéder à un paramètre d'une requête GET en utilisant la méthode get(). Exemple: {{app.request.query.get("nom_parametre")}}
- **app.request.server**: Retourne la séquence de la variable global $_SERVER de PHP. Par exemple l'exemple suivant retourne le nom du serveur hôte qui exécute le script. Exemple:

  {{ app.request.server.get("SERVER_NAME") }}

- **app.request.parameter**
  Permet d'accéder à un paramètre d'une requête POST en utilisant la méthode get(). Exemple: {{app.request.parameter.get("nom_parametre")}}

3. La Syntaxe de base

g.Les variables globales

- **app.request.cookies**
  Permet d'accéder à un paramètre contenu dans un COOKIE en utilisant la méthode get(). Exemple: {{app.request.cookies.get("nom_parametre")}}
- **app.request.headers**
  Retourne toutes les informations du header de la requête HTTP, permet notamment de récupérer le user-agent, le referer, etc ...
- **app.request.content**
  Retourne toutes les informations du contenu de la requête HTTP.
- **app.request.languages**
  Permet de récupérer la séquence des langages acceptés par le navigateur, par exemple : fr, fr-FR, etc.

3. La Syntaxe de base

g.Les variables globales

- **app.request.charsets**
  Permet de récupérer la séquence des jeux de caractères acceptés par le navigateur, par exemple : ISO-8859-1, UTF-8, etc.
- **app.request.acceptableContentTypes**
  Permet de récupérer la séquence des types de contenus acceptés par le navigateur, par exemple : text/html, application/xml, etc.
- **app.request.pathInfo**
  Renvoie les informations sur le chemin d'accès de l'application sans le nom de domaine.
- **app.request.requestUri**
  Retourne l'uri qui est le chemin de la page courante sans le nom de domaine.

3. La Syntaxe de base

g.Les variables globales

- **app.request.baseUrl**
  Retourne l'url de base de l'application.
- **app.request.basePath**
  Renvoie le path de la base de l'application.
- **app.request.method**
  Permet de récupérer les méthodes de requêtes qui ont été utilisés à l'appel de la page, comme par exemple : POST, GET, etc.
- **app.session**
  Permet d'accéder à un paramètre contenu dans une SESSION en utilisant la méthode get(). Exemple: {{app.request.session.get("nom_parametre")}}

3. La Syntaxe de base

h.Les liens

Path() et url ()

- Elles permettent de référencer une route.
- **Path()** genére une URL relative.
- **url()** genére une URL absolue.
- Exemple:

```
<a href="{{ path('produit_route') }}">Produit</a>
<a href="{{ url('produit_route') }}">Produit</a>
```

- On peut également définir une route paramétrée
- `<a href="{{ path('produit_route',{'id': 'value'})}}">Produit détails</a>`

4. Ajout de fichier

Asset(): Permet d'appeler les fichiers ressources css, js,images définis dans le dossier public

- **css**

```
<link  rel= "stylesheet" href="{{asset(css/style.css)}}", type = "text/css">
```

- **JS**

```
<script src="{{ asset('js/script.js') }}"></script>
```

- **Image**

```
<img src="{{ asset ('images/example.jpg') }}" alt="Example"/>
```

• References:

- https://twig.symfony.com/
- https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-199/Moteur-de-Template-Twig-prise-en-main
- https://twig.symfony.com/doc/2.x/deprecated.html