Architectures et technologies WEB

Introduction aux technologies WEB

V01.00 17/08/2016 - MAURICE

Déroulement du cours

- *Chapitre 1 : Préambule*
- *Chapitre 1 : Les technologies WEB*
- *Chapitre 2 : La Programmation cliente*
- **Chapitre 3 : Le modèle MVC / Les templates**
- Chapitre 4 : La sécurité
- Chapitre 5 : Les bases de PHP
- Chapitre 6 : PHP 2ème partie / ORM
- Chapitre 7 : SOA / ROA
- Chapitre 8 : Le référencement

Plan de la séance

- Chapitre 3 : **Le modèle MVC / Les templates**
  - Architecture - Le pattern MVC
  - Le gestionnaire de dépendances
  - Le Templating
  - TWIG
  - Exercices

Architecture – Le pattern MVC

Le modèle MVC

- Le Modèle-Vue-Contrôleur (MVC) est une architecture et une méthode de conception qui organise l'interface homme-machine (IHM) d'une application logicielle.
- Ce paradigme divise l'IHM en :
  - un modèle (modèle de données),
  - une vue (présentation, interface utilisateur)
  - un contrôleur (logique de contrôle, gestion des événements, synchronisation),

- Chacun ayant un rôle précis dans l'interface.
- Cette méthode a été mise au point en 1979 par Trygve Reenskaug

- *source wikipedia

Le modèle MVC

- Le modèle représente le comportement de l'application :
- traitements des données, interactions avec la base de données, etc.
- Il décrit ou contient les données manipulées par l'application.
- Il assure la gestion de ces données et garantit leur intégrité.
- Le modèle offre des méthodes pour mettre à jour les données (insertion, suppression, changement de valeur).
- Il offre aussi des méthodes pour récupérer ces données. Les résultats renvoyés par le modèle sont dénués de toute présentation.

Le modèle MVC

- La vue correspond à l'interface avec laquelle l'utilisateur interagit.
- Elle permet :
  - de présenter les résultats renvoyés par le modèle.
  - de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, etc).
- Ces différents événements sont envoyés au contrôleur.
- La vue n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle et d'interagir avec l'utilisateur.
- Plusieurs vues, partielles ou non, peuvent afficher des informations d'un même modèle.

Le modèle MVC

- Le contrôleur prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser.
- Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer.
- Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle, ce dernier avertit la vue que les données ont changé pour qu'elle se mette à jour.

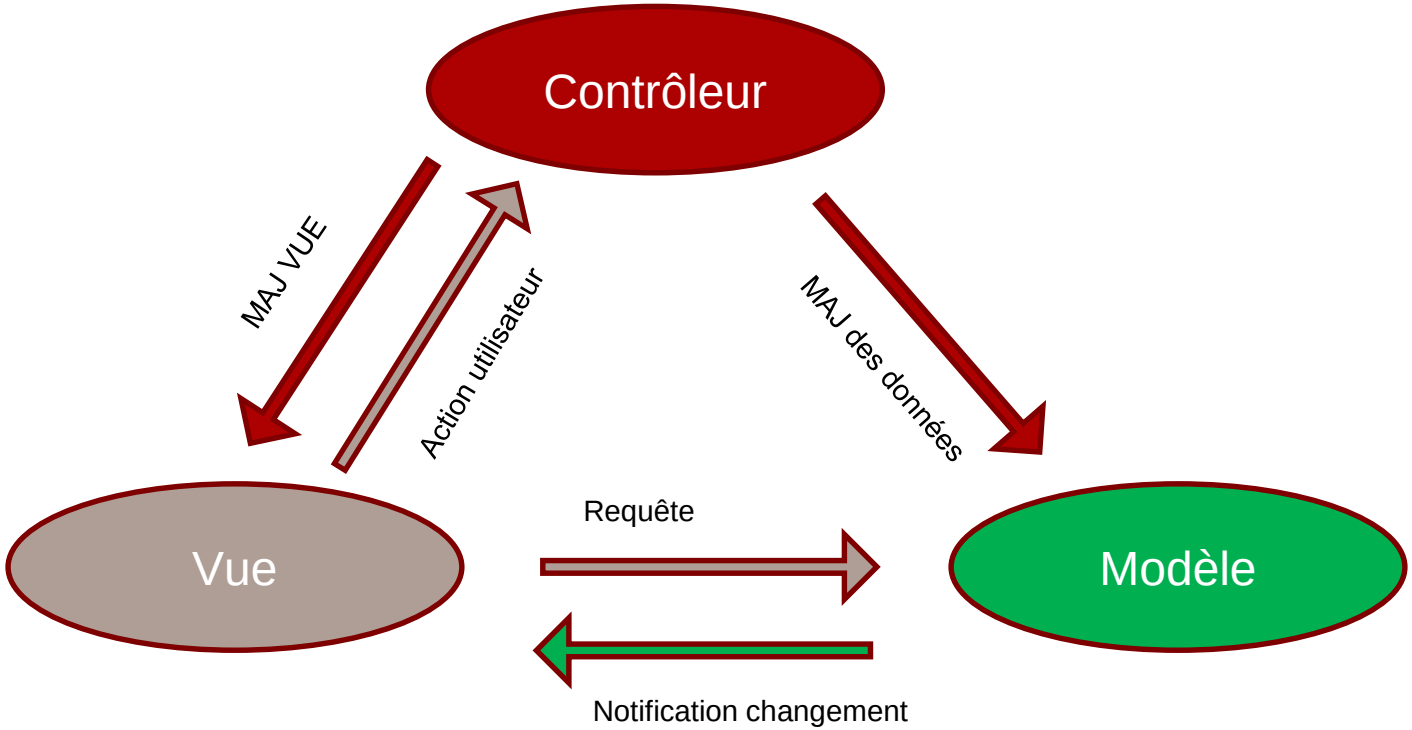
Le modèle MVC

- Certains événements de l'utilisateur ne concernent pas les données mais la vue.
- Dans ce cas, le contrôleur demande à la vue de se modifier.
- Le contrôleur n'effectue aucun traitement, ne modifie aucune donnée. Il analyse la requête du client et se contente d'appeler le modèle adéquat et de renvoyer la vue correspondant à la demande.

Le modèle MVC

- la requête envoyée depuis la vue est analysée par le contrôleur (par exemple un clic de souris pour lancer un traitement de données),
- le contrôleur demande au modèle approprié d'effectuer les traitements et notifie la vue que la requête est traitée (via par exemple un handler ou callback),
- la vue notifiée fait une requête au modèle pour se mettre à jour (par exemple affiche le résultat du traitement via le modèle).

Le modèle MVC



Le modèle MVC

- Symfony est un framework MVC libre écrit en PHP 5.
- le site du framework Symfony a été lancé en octobre 2005.
- À l'origine du projet, on trouve une agence web française, Sensio, qui a développé ce qui s'appelait à l'époque Sensio Framework1 pour ses propres besoins et a ensuite souhaité en partager le code avec la communauté des développeurs PHP.

Le modèle MVC

- La nouvelle branche 2.0 (mars 2011) offre des gains de performances.
- L'utilisation d'un framework comme symfony apporte :
  - des garanties sur la sécurité (XSS, CSRF, SQL Injection...)
  - un web debug
  - des bonnes pratiques de développement

- http://www.symfony-project.org/

- *source wikipedia

La gestion des dépendances

Les dépendances

- Composer est un outil de gestion des dépendances
- Nécessite au minimum PHP 5.3.2
- Sous licence MIT
- https://getcomposer.org/

Les dépendances

- Gère les déclarations des dépendances d'un projet
- Gère les dépendances de librairies
- Gère les dépendances en cascade
- Installe et mets à jour les dépendances
- Télécharge automatiquement les dépendances

Les dépendances

- Dans le fichier php.ini

  activer la ligne : extension=php_openssl.dll

- Télécharger et installer composer

  https://getcomposer.org/download/

Les dépendances

Générer le fichier : composer.phar

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

Créer le fichier : composer.sh

```
php composer.phar $*
```

Copier les fichiers dans votre répertoire de travail :

- composer.phar
- composer.sh

Les dépendances

Créer le fichier composer.json

```
{
    "name": "vendor/package",
    "description": "Description project",
    "license": "GPL3.0+",
    "authors": [
        { "name": "User Name",
          "email": "user@example.com" }],
    "require": { }
}
```

Les dépendances

Créer le fichier composer.json

```
{     "require": {
     "twig/twig":"~1.0",
     "slim/slim" : "^3.0",
     "slim/twig-view" : "*",
},
"autoload": {
     "classmap": ["src/"],
     "files" : ["php/client.php","php/accueil.php"]
}}
```

Lancer l'installation

`composer install` (déploiement se base sur composer.lock)
`composer update` (créé composer.lock et met à jour les dépedances)
`composer require` (ajoute une dépendance)
`composer remove` (supprimer une dépendance)

Les dépendances

- Package Version

EXACT VERSION
"require": { "vendor/project": "1.2.3"
}

WILDCARD
"require": { "vendor/project":
"1.2.*" }

Equivalent : ***>=1.2***, ***<1.3***

RANGE
"require": { "vendor/project": ">=1.2" }

Valeurs valides : **>, >=, <, <=, *!=***

TILDE
"require": { "vendor/project":
"~1.2.3" }

Equivalent : **>=1.2.3**, **<1.3**

Les dépendances

- Composer génère un fichier `autoload` permettant de charger toutes les dépendances de l'application en un seul `require`.

```
<?
    require 'vendor/autoload.php';

?>
```

Le templating

Les vues

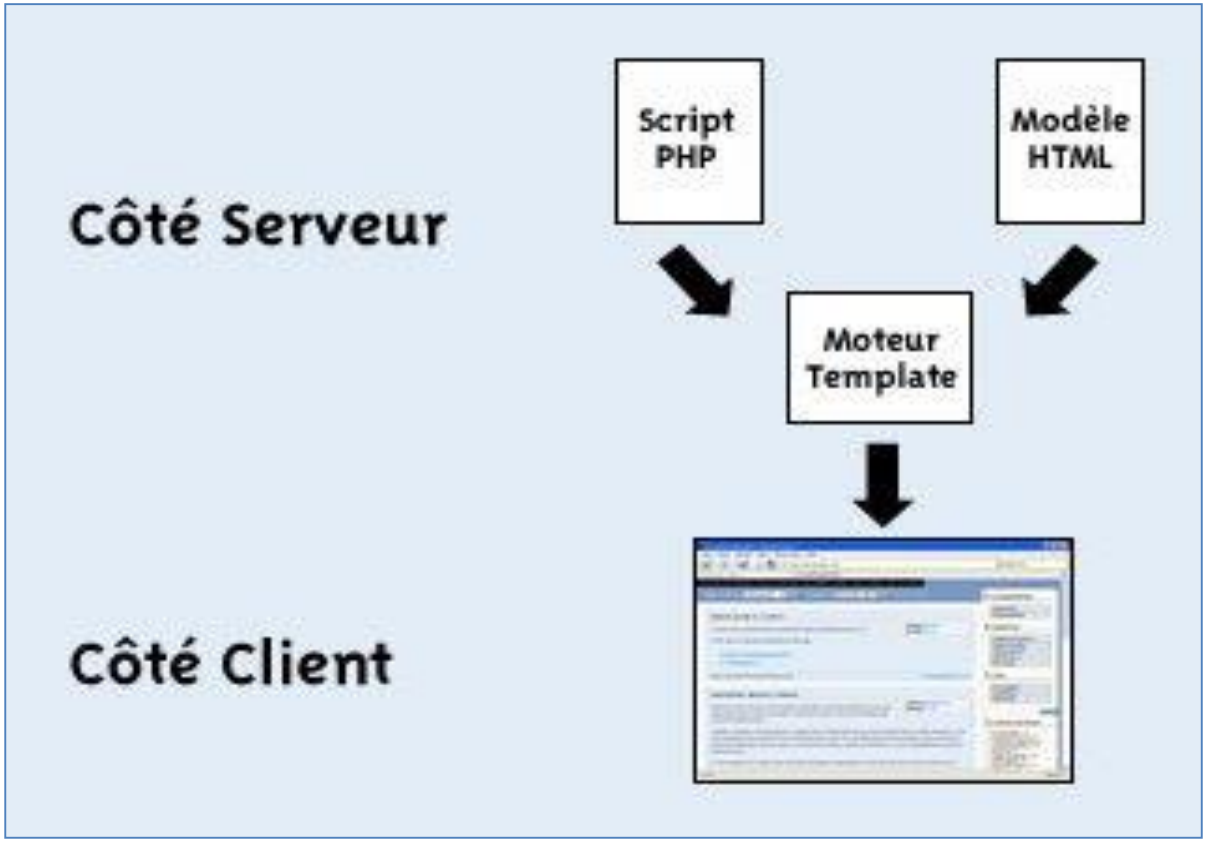
Le templating

- Le principe est de séparer le code de la mise en page html.
- Le code ou la logique applicative est placé dans un fichier, la mise en page contenant les balises html dans un autre fichier.
- le programmeur se charge de la partie scripting, et le designer de la mise en page.

Le templating

- L'avantage évident est de pouvoir travailler uniquement sur la mise en page, sans modifier le code et inversement, ou de répartir efficacement le travail.
- Simplification du code, les sources sont plus clairs, plus lisibles, plus faciles à maintenir.
- Gain de productivité avec la possibilité de dériver rapidement de nouveaux écrans,
- Homogénéité de l'interface assurée par des gabarits génériques à toutes l'application.

Le templating



Définition (1/3)

- Les systèmes de template ont pour objectif de permettre à une équipe de développement de séparer la présentation de la logique applicative et du contenu.
- Les modèles (= templates) de document sont écrits en HTML.
- Ils incluent des mots clés qui seront remplacés par le contenu issu de l'exécution d'un code placé dans des scripts à part et utilisant des données pouvant provenir d'ailleurs (une base de données par exemple).

Définition (2/3)

- Le webdesigner peut travailler le code HTML sans ce soucier du PHP.
- Les développeurs peuvent se concentrer sur le code métier sans interférer sur le travail du designer.
- Les scripts PHP deviennent beaucoup plus légers et les changements dans la charte graphique du site en sont grandement simplifiés.

Définition (3/3)

- "Templates" se traduit en français par "gabarits HTML".
- Ils permettent de créer facilement des skins pour vos sites.
- Cette technologie va à contre poids des méthodes de développement traditionnelles où le et le code HTML sont mélangés, cela nuisant fortement à la lisibilité et à la maintenabilité du site.

Lesmoteurs de templates

- RAINTPL : http://www.raintpl.com/

- LiteTemplate
  - un moteur de template simple, programmé en php4. Il permet de séparer le code php de la mise en page HTML.
  - Le site officiel est http://sourceforge.net/projects/litetemplate et comprend une documentation tres complet faites d'exemples et la classe bien sur.

- ModeliXe
  - http://www.modelixe.org/
  - ModeliXe est un moteur de templates écrit en PHP. Les graphistes apprécieront son support dans Dreamweaver, et sa compatibilité XML.

- DWOO : http://dwoo.org/
  - Moteur de Template supportant l'héritage
  - Connu pour sa rapidité
- PHPTAL : http://phptal.org/
- FastTemplate : http://www.thewebmasters.net/
- MALA Template : http://www.mala-template.net/
- Vtemplate : http://vtemplate.sourceforge.net/
- Templeet : http://www.templeet.org/
- TinyButStrong : http://www.tinybutstrong.com/

SMARTY

Template PHP

SMARTY (1/2)

- http://www.smarty.net/
- Dernière version v 3.1.x
- Sous licence : GNU LESSER GENERAL PUBLIC LICENSE, v2.1

- SMARTY est très rapide
- Pas d'analyse de template coûteuse, une seule compilation
- SMARTY sait recompiler que les fichiers de templates qui ont été modifiés
- Le langage de template est extrémement extensible.

SMARTY (2/2)

- Syntaxe des templates configurable :
- vous pouvez utiliser {}, {{}}, <!--{}-->, comme délimiteurs

- Il n’est pas possible d’inclure du code PHP directement dans vos templates,
- Support de cache intégré
- Fonctions de gestion de cache personnalisables
- Architecture de plugins

Exemple

•Exemple de template (index.tpl)

```
{* Smarty *}
<h1>Hello, {$name}!</h1>
```

•index.php

```
// charge la bibliothèque Smarty
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->template_dir = '/librairie/templates/';
$smarty->compile_dir = '/librairie/templates_c/';
$smarty->config_dir = '/librairie/configs/';
$smarty->cache_dir = '/librairie/cache/';
$smarty->assign('name','Ned');
$smarty->display('index.tpl');
```

Exemple

- Commentaires

```
{* Smarty *}
```

- Inclusion de fichiers

```
{include file="header.tpl"}
```

- Utilisation de variables ds un TPL

```
{$Nom}
```

- Assignation de variables en PHP

```
$smarty->assign("nom","Jacques");
$smarty->assign("noms",array("John","Mary","James"));
```

- Génération de la page à partir d'un gabarit

```
$smarty->display('index.tpl');
```

TWIG

- Twig est un moteur de templates pour le langage de programmation PHP, utilisé par défaut par le framework Symfony.
- Il aurait été inspiré par Jinja, moteur de template Python2.
- Twig est Open Source sous licence BSD
- Ses principales fonctionnalités sont :
  - contrôle de flux complexe
  - échappement automatique
  - héritage des templates
  - filtres variables
  - internationalisation

TWIG

- Installation via composer

```
composer require twig/twig:~2.0
```

Les Templates

- Les fichiers Templates sont des pages HTML
- Ils ont par défaut l'extension : « .twig »
- Ils contiennent donc avant tout du code HTML, du CSS et du Javascript,…
- Ils sont dynamiques et des zones variables peuvent être intégrer spécifiquement dans les parties réservées de ces pages.
- Ils peuvent être structurés et inclurent des sous

```
// Localisation des Templates
$loader = new Twig_Loader_Filesystem($templateDir);
// ou via tableau de répertoires
$loader = new Twig_Loader_Filesystem(array($templateDir1,
$templateDir2));
```

Les Templates

- Pour démarrer Twig, il est nécessaire de créer un environnement en définissant quelques options :
  - cache : dossier utilisé par Twig pour mettre en cache les pages PHP générées.
  - charset : par défaut à utf-8, définit l'encodage de votre projet.
  - autoescape : échappe automatiquement les variables. Le code HTML contenu dedans ne sera pas interprété par le navigateur. Banalise les caractères réservés : < >

```
$twig = new Twig_Environment($loader, array(
    'cache' => false
));
```

Les Templates

- Exemple

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="fr">
<head>
<link rel="stylesheet" href="style.css" />

<title>{{titre}}</title>

</head>
<body>
      <!- Contenu WEB -->
</body>
</html>
```

Les Templates

- Génération de la vue par application d'un pattern
- Chargement du pattern à utiliser
- Affectation des valeurs aux zones variables définies dans le pattern
- Envoi du flux générés au client WEB

```
<?php
    $template = $twig->loadTemplate('index.twig');
    echo $template->render(array(
      'titre' => 'Le titre de ma page'
    ));
?>
```

Les Templates

- Quelques bases sur la constructions des templates :
- Les balises Twig sont contenues entre des accolades { }
- 3 types de balises Twig

```
{{ … }} affiche quelque chose ;
{% … %} fait quelque chose ;
{# … #}  n'affiche rien et ne fait rien : Permet d'inclure des commentaires dans vos templates
```

Les Templates

- Affichage d'une variable se fait avec les doubles accolades {{ … }}

```
Login : {{ login }}
Element : {{ montableau['id'] }}
Attrbut : {{ objet.attribut }}
```

Le fonctionnement de la syntaxe {{ objet.attribut }} suit quelques règles dans cet ordre :

Si objet et un tableau : `objet['attribut']`

Si objet est un objet : `objet->attribut.`

Si attribut() est une méthode : `objet->attribut()`

Si getAttribut() est une méthode : `objet->getAttribut()`

Si isAttribut() est une méthode : `objet->isAttribut()`

Sinon `null`

Les Templates

- TWIG permet d'accéder directement à des variables globales à partir des templates :

app.session
Permet d'accéder à un paramètre contenu dans une SESSION
```
{{app.request.session.get("nom_parametre")}}
```

app.request.cookies
Permet d'accéder à un paramètre contenu dans un COOKIE
```
{{app.request.cookies.get("nom_parametre")}}
```

app.request.query
Permet d'accéder à un paramètre d'une requête GET
```
{{app.request.query.get("nom_parametre")}
```

app.request.parameter
Permet d'accéder à un paramètre d'une requête POST
```
{{app.request.parameter.get("nom_parametre")}}
```

Les Templates

- set permet de créer une variable TWIG dans le template :

```
{% set foo = 'bar' %}
```

Les Templates

Les filtres utiles

https://twig.symfony.com/doc/2.x/filters/index.html

- `Upper / lower / capitalize`

Met toutes les lettres en majuscules / minuscules / capitales . `{{ var|upper }} {{ var|lower }} {{ var| capitalize}}`

- `Striptags`

Supprime toutes les balises XML. `{{ var|striptags }}`

- `date`

Formate la date selon le format donné en argument. La variable en entrée doit être une instance deDatetime.

`{{ date|date('d/m/Y') }}`

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

- `format`

Insère des variables dans un texte

`{{ "Il y a %s pommes et %s poires"|format(153, nb_poires) }}`

- **`length`**

Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne.

Longueur de la variable : `{{ texte|length }}`

Nombre d'éléments du tableau : `{{ tableau|length }}`

Les Templates

- Les structures de contrôles : les conditionnels

```
{% 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 %}



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

Les Templates

- Les structures de contrôles : les itérations

```
<ul>
  {% for membre in liste_membres %}
    <li>{{ membre.pseudo }}</li>
  {% else %}
    <li>Pas d'utilisateur trouvé.</li>
  {% endfor %}
</ul>
```

Les Templates

- `{{ loop.index }}` : Le numéro de l'itération courante (en commençant par 1).
- `{{ loop.index0 }}` : Le numéro de l'itération courante (en commençant par 0).
- `{{ loop.revindex }}` : Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 1).
- `{{ loop.revindex0 }}` :Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 0).
- `{{ loop.first }}` true si c'est la première itération,false sinon.
- `{{ loop.last }}` true si c'est la dernière itération,false sinon.
- `{{ loop.length }}` Le nombre total d'itérations.

Les Templates

- https://twig.symfony.com/doc/2.x/tags/for.html

```
// Boucle avec un incrément
{% for i in 0..10 %} * {{ i }} {% endfor %}

// Boucle sur une liste de caractères
{% for letter in 'a'..'z' %} * {{ letter }} {% endfor %}

// Boucle sur une liste de caractères filtrés
{% for letter in 'a'|upper..'z'|upper %} * {{ letter }} {%
endfor %}

// Boucle conditionnelle
{% for user in users if user.active %} {{ user.username|e
}} {% endfor %}
```

Les Templates

- Les inclusions de template

```
{% include 'header.html' %}
Body
{% include 'footer.html' %}
```

- Les inclusions de template avec communications de variables

```
{% include 'template.html' with {'foo': 'bar'} %
```

Les Templates

- La notion de bloc est liée au concept d'héritage de template
- Elle permet d'étendre un template parent

```
<!DOCTYPE html>
<html>
<head>
{% block head %}
<link rel="stylesheet" href="style.css" />
<title>{% block title %}{% endblock %} - My Webpage</title>
{% endblock %}
</head>
<body>
<div id="content">{% block content %}{% endblock %}</div>
</body> </html>
```

Les Templates

- Le template enfant doit redéfinir le contenu des blocs contenus dans le template parent

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}


{% block content %} Bonjour {% endblock %}
```

Les Templates

- vous pouvez utiliser{{ parent() }} pour obtenir le contenu par défaut du bloc parent.
- Et compléter le contenu du bloc parent par un contenu spécifique

```
{% block head %} {{ parent() }}

<style type="text/css"> .important { color: #336699; }
</style>
{% endblock %}
```

Bonnes Pratiques

- Toujours séparer le code du rendu.
- Définir les structures d'échanges entre le code PHP et le template. Travail de spécification technique à réaliser
- Eviter le PHP dans le template
- Bien définir les différents zonning de l'application
- Structurer les template en block pouvant être enrichis par héritage. Se limiter à 3 niveaux !
- Définir un block contenant la structure de base d'un page HTML contenant le sequelette de la page, le DOCTYPE, l'encodage, les CSS, les JS.