



Symfony 6.2

Nidhal Chérif



Symfony

Prérequis

- PHP/Programmation Orientée Objet
- Les espaces de nom (namespace)
- Gérer le code sur Git/Github
- MVC (Modèle/View/Contrôleur)

Plan de la formation

- ☐ PHP POO/PDO-Namespaces- MVC
- ☐ Installation de l'environnement
- ☐ Création d'un projet Symfony
- ☐ Présentation de Symfony et création d'une première page
- ☐ Contrôleur
- ☐ Template : TWIG
- ☐ Doctrine
- ☐ Formulaire
- ☐ Relation ManyToOne, OneToMany, OneToOne et ManyToMany
- ☐ DQL

Les namespaces



Symfony

Problème

- La taille de votre bibliothèque de code PHP augmente
=> risque accru de redéfinition accidentelle d'une fonction ou d'un nom de classe qui a été déclaré auparavant
- On ne peut pas créer deux constantes, deux fonctions ou deux classes portant le même nom.
- Le problème est exacerbant lorsque vous tentez d'ajouter des composants tiers ou plugins

Solution :

- Les problèmes de collision de noms peuvent être résolus avec les espaces de noms.
- Les constantes PHP, classes et fonctions peuvent être regroupées en bibliothèques d'espace de noms.
- Solution apportée avec PHP 5.3

Déclaration :

```
<?php
namespace MonProjet;
// ... code ... ?>
```

✎ Par défaut, toutes les constantes, les classes, et les noms de fonction sont placés dans l'espace global

Exemple :

faire précéder le
nom de la fonction
d'un backslash (\)

```
<?php
namespace Nidhal;
function strtolower()
{   echo 'BONJOUR TOUT LE MONDE'; }
    //Appel de strtolower du namespace Nidhal
strtolower();
    //Appel de la fonction du namespace global
echo \strtolower('BONJOUR TOUT LE MONDE');
?>
```

il est préférable d'utiliser les accolades

```
<?php
namespace A
{ function quelNamespace()
  { echo 'A'; }
  quelNamespace();
}
namespace B
{ function quelNamespace()
  { echo 'B'; }
  quelNamespace();
}
namespace // Le code contenu dans ce namespace fera partie du
           //namespace global.
{   echo strlen('Hello world !');
}
?>
```

- PHP vous permet de définir une hiérarchie d'espaces de noms afin de subdiviser vos bibliothèques. Les sous-espaces de noms sont séparés par une barre oblique inverse (\), par exemple :
 - MonProjet \ Nom
 - MonProjet \ Database \ MySQL
 - NomEntreprise \ MonProjet \ Bibliotheque \ FichiersCommuns \ widget

Exemple : Script du fichier « **library1.php** »

```
<?php
namespace App\Lib1;
const MACONST = `L2DSI`;
function Afficher_msg()
    { return " Bonjour tout le monde "; }

class MaClasse
{ function QuiSuisJe()
    { return " Je suis une fonction ";}
} ?>
```


- Inclusion du code précédent dans un autre fichier « MyApp.php »

```
//MyApp.php
<?php
require 'Library1.php';
use App\lib1;
echo App\lib1 \MACONST."<br/>";
echo App\lib1 \Afficher_msg()."<br/>";
$obj = new App\lib1 \MaClasse();
$obj->QuiSuisJe();
echo '<br/>' ; ?>
```

- Utilisation d'un alias

```
//MyApp.php
<?php
require 'Library1.php';
use App\lib1 as A ;
echo A\MACONST."<br/>";
echo A\MaFonction()."<br/>";
$obj = new A\MaClasse();
$obj-> QuiSuisJe();
echo '<br/>' ; ?>
```

Importer des classes

```
//MyApp.php
<?php
require 'Library1.php';
use App\lib1 \MaClasse;
$obj = new MaClasse(); // Se transforme en $obj = new lib1\APP1\MaClasse.
$obj-> QuiSuisJe();
echo '<br/>' ; ?>
```

uniquement des classes ça ne fonctionnera pas avec des constantes ou fonctions !!!

- Inclusion du code précédent dans un autre fichier « MyApp.php »

```
//MyApp.php
<?php
require 'Library1.php';
use App\lib1;
echo App\lib1 \MACONST."<br/>";
echo App\lib1 \Afficher_msg()."<br/>";
$obj = new App\lib1 \MaClasse();
$obj->QuiSuisJe();
echo '<br/>' ; ?>
```

- Utilisation d'un alias

```
//MyApp.php
<?php
require 'Library1.php';
use App\lib1 as A ;
echo A\MACONST."<br/>";
echo A\MaFonction()."<br/>";
$obj = new A\MaClasse();
$obj-> QuiSuisJe();
echo '<br/>' ; ?>
```

Importer des classes

```
//MyApp.php
<?php
require 'Library1.php';
use App\lib1 \MaClasse;
$obj = new MaClasse(); // Se transforme en $obj = new lib1\APP1\MaClasse.
$obj-> QuiSuisJe();
echo '<br/>' ; ?>
```

uniquement des classes ça ne fonctionnera pas avec des constantes ou fonctions !!!

MVC



Symfony

- La partie serveur est coupée en trois morceaux :
 - le modèle qui contient les données (base de données)
 - le contrôleur qui prend les décisions
 - la vue (templates) qui organise la présentation (affichage)



Démo

Introduction

| Qu'est ce que Symfony et pourquoi l'utiliser?

Nidhal CHERIF



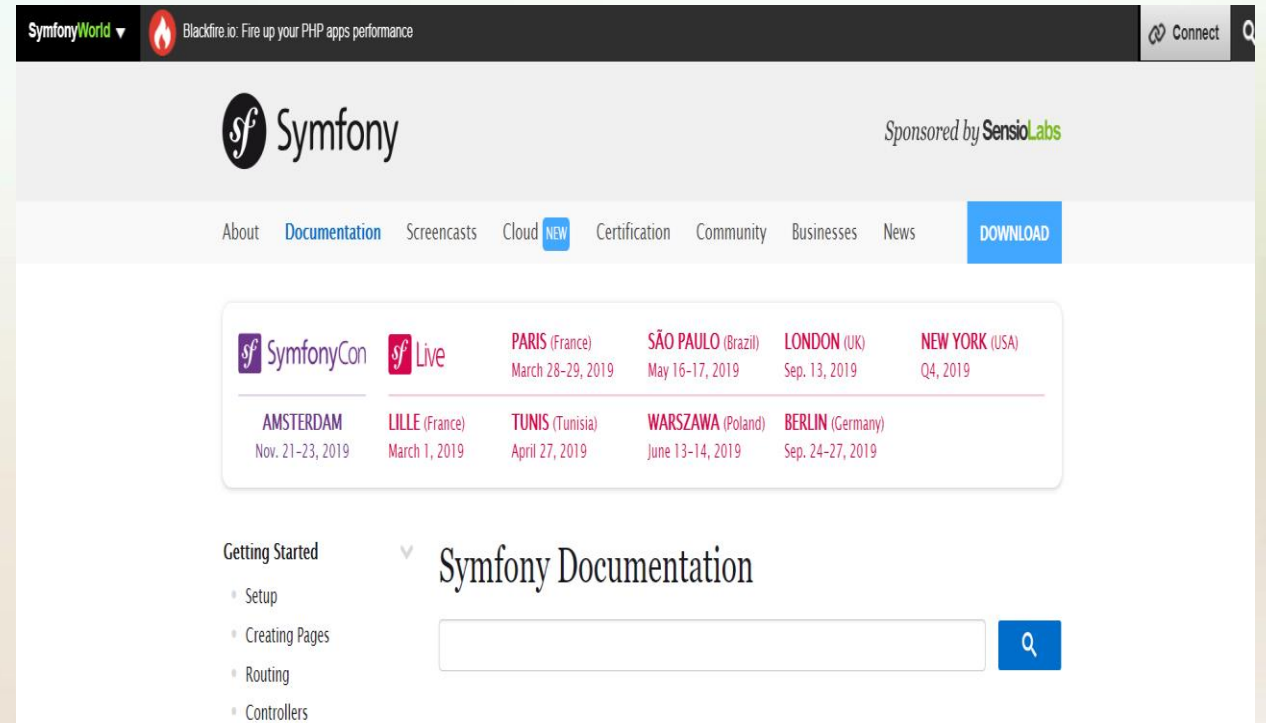
Symfony

- Un Framework = Cadre de travail
un socle de fonctionnement pour travailler simplement
- Framework open source côté serveur basé sur PHP

- Créateur :

The logo for Sensio Labs, featuring the word "Sensio" in black and "Labs" in green.

- Site officiel : <https://symfony.com/doc/>



Un ensemble d'outils déjà
codé



Rapidité de mise en place

Des fonctionnalités :
testées et approuvées

Grande communauté

Documentation riche

- Un ensemble de composants
- Un framework qui facilite la vie
- Une communauté grandissante (Sensiolab) et documentation riche
- Adapté aux grandes et aux petites applications
- Utilise une Architecture MVC 2 (Modèle- vue- Contrôleur)

- Développer plus vite
- Faciliter le travail en équipe
- sécurité
- Simplifier la maintenances et l'évolution
- Se concentrer sur la logique métier
- Ne pas réinventer la roue!

The logo for L'EXPRESS, featuring the word "L'EXPRESS" in white, bold, sans-serif capital letters on a solid red rectangular background.The logo for 20minutes.fr, featuring the number "20" in a stylized blue font with a white dot, followed by the text "minutes.fr" in a blue serif font.The Yahoo! logo, consisting of a purple circle containing a white "Y", followed by a purple exclamation mark and the word "YAHOO!" in a purple, bold, sans-serif font.The TF1 logo, featuring the letters "T" and "F" in white on a blue background, and the number "1" in white on a red background.The OpenClassrooms logo, featuring the word "OPENCLASSROOMS" in a blue, bold, sans-serif font.The Dailymotion logo, featuring the word "Dailymotion" in a blue, bold, sans-serif font.The BNP Paribas logo, featuring a circular arrangement of five teal stars of varying sizes, with the text "BNP PARIBAS" in a black, bold, sans-serif font to the right.

Plan

- ❑ Installation et organisation
- ❑ Contrôleur
- ❑ Template
- ❑ Doctrine: Base de données
- ❑ Formulaire
- ❑ Relation ManyToOne, OneToMany, OneToOne et ManyToMany

Installation



Symfony

- Installer PHP 8.1 ou supérieur (Xampp)
- Configurer PHP sous Windows
- Installer Composer 
- Installer Symfony cli : <https://symfony.com/download>
- Installer git : <http://git-scm.com/download/win>

- Deux solutions pour créer un projet complet My_Project:

- avec symfony

```
>symfony new my_project_directory --version="6.2.*" --webapp
```

- avec composer :

```
>composer create-project symfony/skeleton:"6.2.*" my_project_directory
```

```
>cd my_project_directory
```

```
>composer require webapp
```

- Visual studio code :
Ajouter les extensions :
 - **Extension PHP DocBlocker**
 - **Extension Twig**
 - **Extension PHP Namespace Resolve**
- PHP Storm :
installer le plugging :
 - **Symfony**
 - **Emmet Everywhere**

Organisation du code



Symfony

■ Contrôleurs en PHP :

- récupération des données, traitement et envoi aux vues
- gestion des url par annotations

■ Vues en Twig :

- langage de Templates simple et puissant
- une vue par « type de page »
- héritage

■ Modèle en MySQL :

- utilisation de Doctrine (ORM)
- Objet

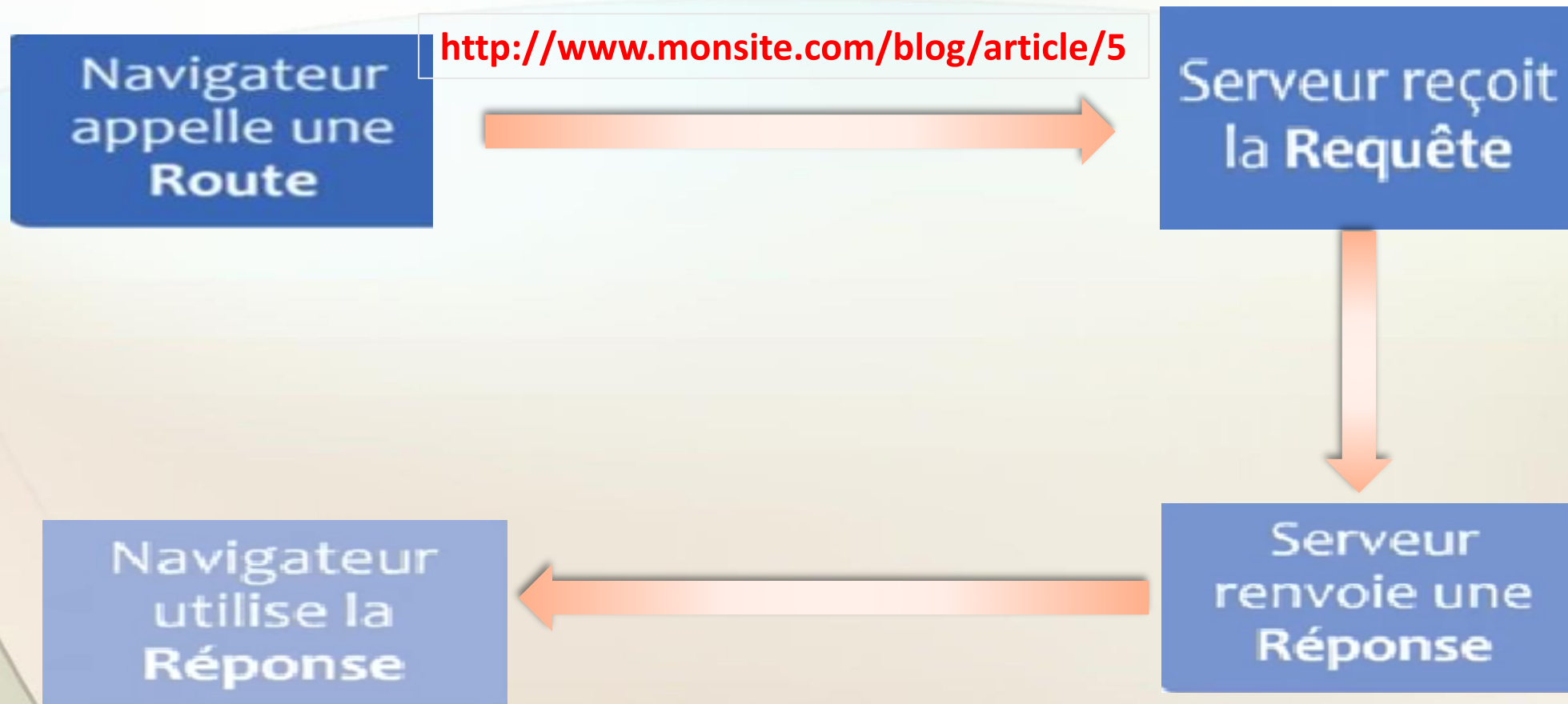
- Trois répertoires pour développer : **src** (code PHP) , **templates** et **config** (configuration)
 - **src/Controller** : les contrôleurs
 - **src/Entity** : les classes pour la base de données
 - **Templates**: les vues (templates) •
 - **config** : les fichiers de configuration
- Autres répertoires :
 - **public** : contient les fichiers publiquement accessibles (images, CSS, etc.)
 - **bin** : contient les exécutables (notamment console)
 - **var** : cache, logs, etc.
 - **tests** : pour les tests automatiques
 - **vendor** : bibliothèques

Controller



Symfony

- Le contrôleur est la « glu » d'une application Symfony ;
- Il « utilise » tous les autres composants (base de données, formulaires, templates, etc.) pour générer la réponse suite à une requête ;
- Répertoire : **src/App/Controller**
- **Commande: >php bin/console make:controller**



Exercice : Hello World

- Créer un contrôleur HelloController
- Et une méthode hello1 et une route en yaml puis en annotation permettant d'afficher « hello world » ou « hello prénom » sachant que prénom est variable.

Exigences :

- **/hello** doit afficher "Hello World"
- **/hello/Omar** doit afficher "Hello Omar"
- **/hello/Ines** doit afficher "Hello Ines«
- **/hello/111** OU **hello/1omar** sont interdits

Notion de service dans Symfony



Symfony



50%

Le
Container
de services

50%

**Routes
Request
Response
Controllers**

SERVICE = CLASSE

SERVICE = OUTIL



```
> php bin/console debug:autowiring
```

- Sous **src** créer un dossier **taxes**
- Sous **taxes** Créer un fichier **calculator.php**
- Créer un espace de nom `App\taxes`
- une classe **calculator** avec une méthode qui calcule le montant TVA d'un prix donné en paramètre
- Calculator est devenu maintenant un service centralisé
- Tester le avec :
 - `> php bin/console debug:autowiring --all`

Twig



Symfony

Simple

- Écriture facile des affichages : `{{var}}`
- Apporte beaucoup de fonctionnalités : filtre, héritage du layout....

Sécurisé

- les données affichées sont immunisées contre les éventuelles attaques malveillant : comme XSS(cross-site scripting)

Absence de php

- Simplifier le travail d'un intégrateur

- Avec PHP:

```
<p class="date">  
  < ?php echo $MaDate ; ?>  
</p>
```

- Avec Twig :

```
<p class="date"> {{ MaDate }}</p>
```

- **{{ maVar }}** : Les doubles accolades permettent d'imprimer une valeur, le résultat d'une fonction...
- **{% for page in arrPages %}... {% endfor %}** : Les accolades pourcentage permettent d'exécuter une fonction, définir un bloc...
- **{# Les commentaires #}**: syntaxe pour les commentaires

Description	Exemple Twig	Équivalent PHP
Afficher une variable pseudo	<code>{{ pseudo }}</code>	<code><?php echo \$pseudo; ?></code>
Afficher l'index d'un tableau	Identifiant : <code>{{ user['id'] }}</code>	Identifiant : <code><?php echo \$user['id']; ?></code>
Afficher l'attribut d'un objet	Identifiant : <code>{{ user.id }}</code>	Identifiant : <code><?php echo \$user->getId(); ?></code>
Afficher une variable en lui appliquant un filtre. Ici, « upper » met tout en majuscules :	Pseudo en majuscules : <code>{{ pseudo upper }}</code>	Pseudo en lettre majuscules : <code><?php echo strtoupper(\$pseudo); ?></code>
Afficher une variable en combinant les filtres. « striptags » supprime les balises HTML. « title » met la première lettre de chaque mot en majuscule.	Message : <code>{{ news.texte striptags title }}</code>	Message : <code><?php echo ucwords(strip_tags(\$news->getTexte())); ?></code>
Utiliser un filtre avec des arguments.	Date : <code>{{ date date('d/m/Y') }}</code>	Date : <code><?php echo \$date->format('d/m/Y'); ?></code>
Concaténer	Identité : <code>{{ nom ~ " " ~ prenom }}</code>	Identité : <code><?php echo \$nom.' '.\$prenom; ?></code>

❑ Syntaxe en Twig :

```
{% if condition1 %}  
    Traitement 1.....  
{% else if condition2 %}  
    Traitement 2 ....  
{% else %}  
    Traitement 3 .....  
{% endif %}
```

❑ Equivalent en PHP :

```
if ( condition1)  
    { Traitement 1.....}  
else if (condition2 )  
    { Traitement 2 ....}  
else  
    { Traitement 3 .....}
```

- **Syntaxe en Twig :**

```
{% for valeur in tableau %}  
    {{valeur}} </br>  
{% endfor %}
```

- **Equivalent en PHP :**

```
foreach( $tableau as $valeur)  
    {echo $valeur ; }
```

- Parcourir un tableau associatif

- **Syntaxe en Twig :**

```
{% for clef,valeur in tableau %}  
  {{clef}}: {{valeur}} <br/>  
{% endfor %}
```

- **Equivalent en PHP :**

```
foreach( $tableau as $clef=> $valeur)  
{echo "$clef : $valeur <BR/>" ; }
```

- Un raccourci très appréciable avec Twig est la possibilité d'afficher un texte alternatif lors d'une tentative d'itération sur un tableau vide :

```
{% for valeur in tableau %}
```

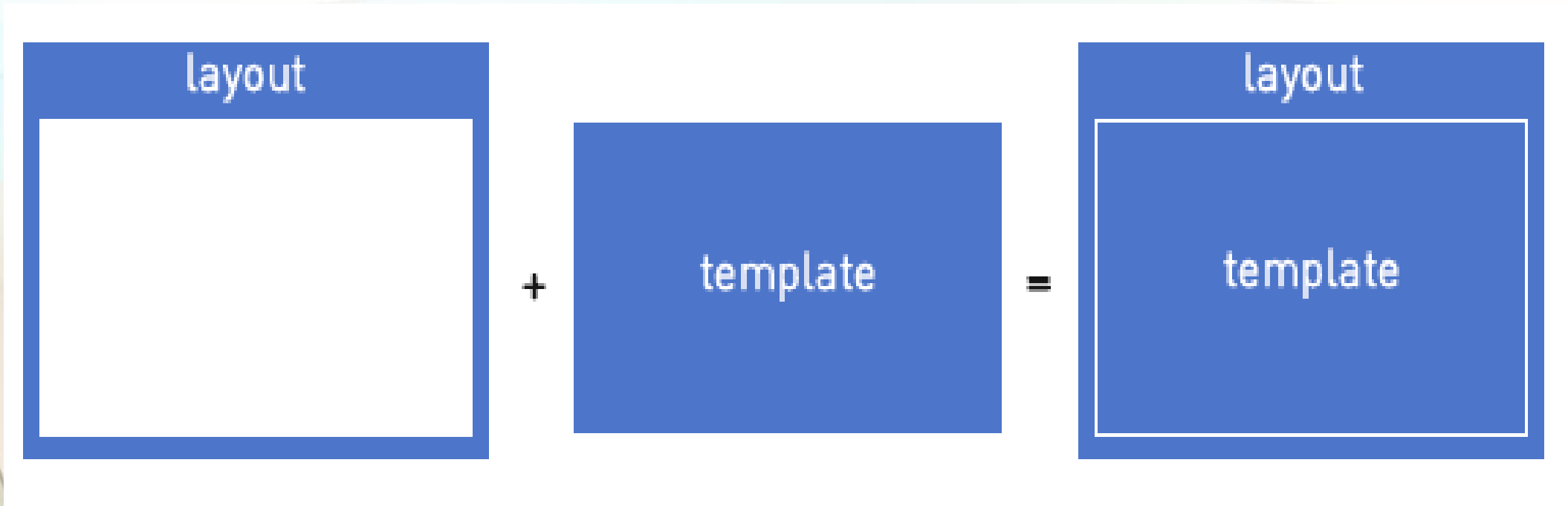
```
    {{valeur}} <br/>
```

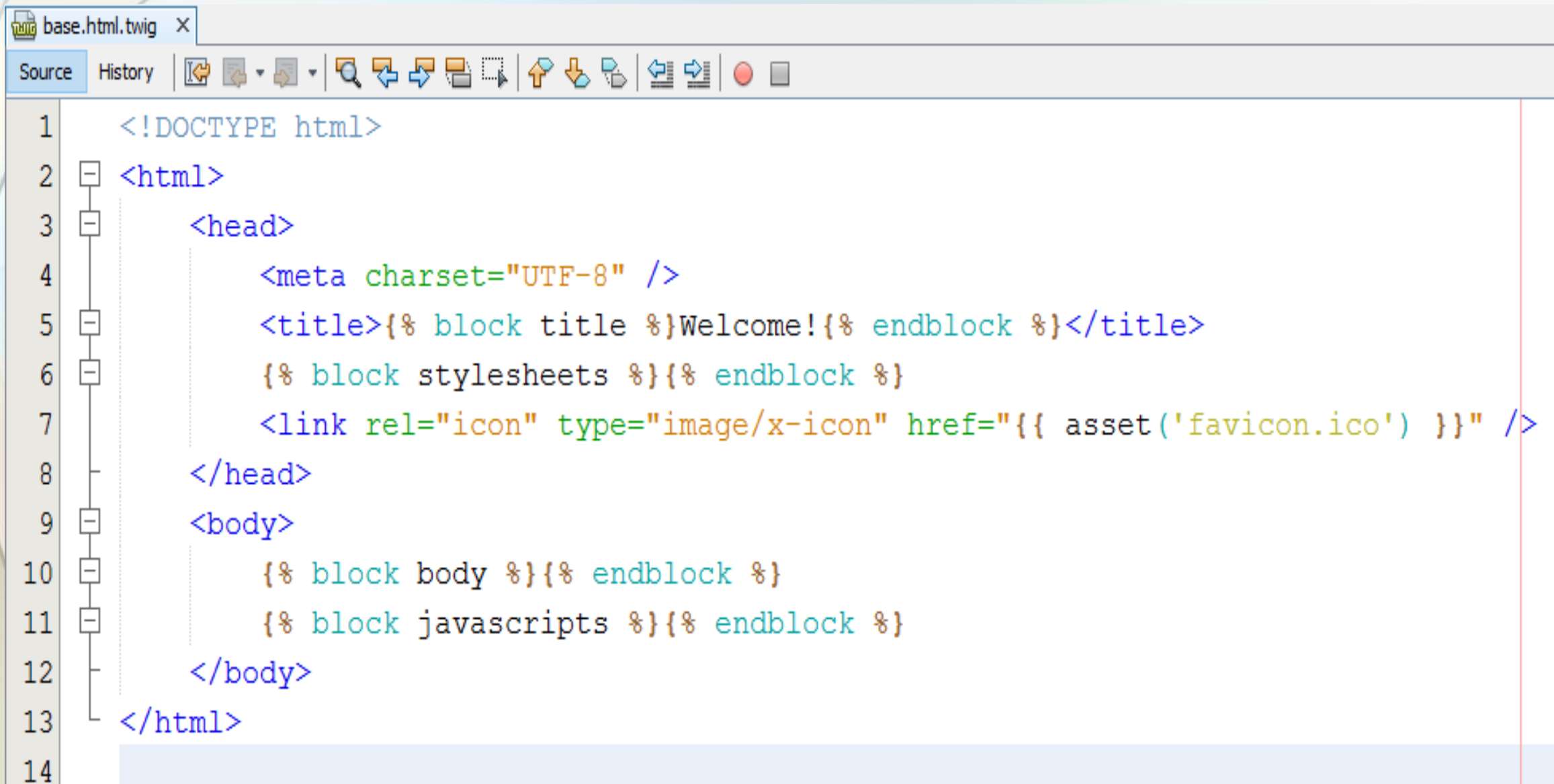
```
{% else %}
```

```
Le tableau est vide. <br/>
```

```
{% endfor %}
```

- nous allons utiliser un autre modèle pour résoudre le problème de la mise en page : le **modèle décorateur** il s'agit du **layout**





```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8" />
5     <title>{% block title %}Welcome!{% endblock %}</title>
6     {% block stylesheets %}{% endblock %}
7     <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" />
8   </head>
9   <body>
10    {% block body %}{% endblock %}
11    {% block javascripts %}{% endblock %}
12  </body>
13 </html>
14
```

- Les blocks sont des zones, des marqueurs, placés sur les templates et dont le contenu est optionnel : **title**, **stylesheets**, **body** et **javascripts**
- Les blocks sont utiles lors de l'héritage de Template :
- grâce à la fonction **{{parent()}}** nous demandons à réafficher le contenu du block du template parent et nous y ajoutons ensuite un texte complémentaire

- Il faut spécifier que nous allons utiliser le layout « base.html.twig » dans notre template avec le code suivant :

```
{% extends 'base.html.twig' %}
```

- Il ne reste plus qu'à surcharger les blocks du layout pour faire apparaître notre page d'accueil:

```
{% extends 'base.html.twig' %}
```

```
{% block title %}
```

```
    {{parent()}} page d'index
```

```
{% endblock %}
```

```
{% block body %}
```

```
    Hello tout le monde !
```

```
{% endblock %}
```


- La fonction **asset()** est utilisée référencier les ressources publiques (**images, feuilles de style, Script js..**) depuis un template.
- Toutes ces ressources sont dans le répertoire **public**.

❑ Syntaxe en twig :

```

```

```
{# référence public/img/logi.png#}
```

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

```
{# ' référence public/css/style.css #}
```

- Certaines parties des templates, considérées comme des modèles pouvant se répéter dans plusieurs pages, pouvant être factorisées.
- Pour résoudre ceci, il suffit d'inclure le code du template dont a besoin dans une autre page (Template incluant)

```
{% extends 'base.html.twig' %}
```

```
{% block body %}
```

Contenu de la page.....

```
{% include '::publicite.html.twig' %}
```

```
{% endblock %}
```

- Les routes sont très utilisées dans les Templates et nous savons comment générer des URL depuis le contrôleur. Mais il est fréquent de vouloir en générer depuis les vues et ceci grâce à la fonction **path**

```
<a href="{{path ('ma_route')}}">visiter ma page</a>  
{#avec ma_route est le nom de votre route #}
```

ORM: Doctrine



Symfony

- **symfony/website-skeleton** est livré par défaut avec une librairie Doctrine, permettant de gérer les interactions entre une application et une (ou plusieurs base de données).
- Doctrine = librairie = Bundle
- Doctrine -> gère les bases de données

Service Doctrine : getDoctrine() ou get('Doctrine')



EntityManager

- ✓ Insert
- ✓ Update
- ✓ delete

Repository

- ✓ Select

- Entité = classe + configuration

```
<?php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\LivreRepository")
 */
class Livre
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $titre;
```

- Doctrine se compose de plusieurs couches : DBAL, ORM et Entité :
 - **DBAL** (Database Abstract Layer) : est une couche de plus bas niveau, son rôle est d'envoyer des requêtes vers une base de données et de récupérer les résultats.
 - **Entité** : Les entités sont les classes d'une application ayant été configurées de manière à ce que Doctrine puisse établir une correspondance entre ces dernières et des tables en base de données.
 - **ORM** (Object Relationnel Mapping, en français : lien objet –relation) son principal rôle est de faire une correspondance entre la base de données relationnelles et la programmation orientée objet

- **Exercice : Créez une entité Category**

- Vous allez créer une nouvelle entité (make:entity) qui s'appellera Category.

Exigences :

- L'entité Category aura **un champ *name* de type string, de longueur 255 et non null**
- L'entité Category aura **un champ *slug* de type string, de longueur 255 et non null**
- Vous devez **générer une migration** (make:migration)
- Vous devez **exécuter la migration** (doctrine:migrations:migrate)

Références

- <https://symfony.com/doc/>
- <https://openclassrooms.com/fr/courses/3619856-developpez-votre-site-web-avec-le-framework-symfony>
- <http://php.net/manual/fr/language.oop5.magic.php>