# Symfony : Comprendre le Framework !

### Tout ce qui sera écrit en orange seront des lignes de commande à faire dans le terminal (Windows)

### Quelques commandes pour se déplacer dans le terminal de Windows :

* **cd ou chdir ⬄ Affiche le nom du répertoire courant ou change le répertoire courant**
* **cls ⬄ Efface le contenu de la fenêtre d’invite de commande.**
* **copy ⬄ Copie un fichier d’un emplacement à un autre.**
* **del ⬄ Supprime un ou plusieurs fichiers**
* **dir ⬄Affiche la liste des fichiers et sous-dossiers contenus dans le répertoire.**
* **exit ⬄ Quitte le programme cmd.exe ou le script batch courant.**
* **md ou mkdir ⬄ Crée un répertoire ou un sous-répertoire.**
* **move ⬄ Déplace un fichier d’un dossier à un autre.**
* **rd ou rmdir ⬄ Supprime un répertoire.**
* **ren ou rename ⬄ Change le nom d’un fichier ou d’un ensemble de fichiers.**
* **php -v** ⬄ Permet de voir la version PHP installer sur notre machine
* **composer -V** ⬄ Pour vérifier la version de composer et donc si composer fonctionne bien , qu’il existe et qu’il est bien installer sur la machine.

Une fois que les 2 points précédent soit ok on peut donc commencer à découvrir Symfony.

# Créer une application Symfony avec composer.

Se rendre dans un dossier où l’ont veut créer le projet, on est pas obligé d’aller dans le dossier www de wamp car on ne va pas travailler avec apache pour faire tourner Symfony mais avec un serveur interne.

* **composer create-project symfony/website-skeleton demo**

Create-project ⬄ Créer une structure de dossiers/fichiers à partir d’un modèle.

Symfony/website-skeletin ⬄Modèle d’un projet de site web symfony.

Demo ⬄ Le nom de mon nouveau projet.

Je dois ensuite me rendre (avec le terminal) dans le dossier où j’ai créé mon application.

* **cd demo**

L’ouvrir avec mon éditeur de code (Visuel Studio code)

* **code .**

Une fois ouvert, je ne vais plus utiliser mon invite de commande externe, mais je peux maintenant commencer à utiliser directement au sein même de mon Visuel Studio Code pour taper toute sortes de commande.

Pourquoi toute sortes de commande ? Car Symfony c’est beaucoup de fichiers et beaucoup de codes. On peut tout faire à la main mais les développeurs de Symfony nous ont fournis la CLI ⬄ Command light interface. C’est un petit assistant qu’on peut appeler par des lignes de commandes et qui va nous aider à gérer notre application Symfony.

# Installer un serveur personnaliser pour lancer mon application Symfony

* **composer require server --dev ⬄** Que je ne vais utiliser que lorsque je vais développer, le jour où je vais passer mon site en prod, tu n’auras pas besoin d’installer cette librairie là.

Composer require [xxx]⬄ Permet d’installer une librairie et de l’utiliser dans le projet !

A la fin de l’installation, on va pouvoir lancer notre application dans notre navigateur.

* **php bin/console server:run**

Ça va lancer notre serveur sur localhost :8000

Notre application ne tourne donc pas sur le port 80, le port 80 c’est pour mon apache/wamp si je veux accéder à d’autres projets qui tourne sur wamp par exemple ou à phpmyadmin.

**ATTENTION Dépendamment de la version de Symfony utilisé, la commande précédente affichera une erreur mais on peut toujours lancer le serveur avec la commande suivante :**

**Depuis la version 4.4 de Symfony il faut Aller sur** [**https://symfony.com/download**](https://symfony.com/download) **et télécharger et installer le logiciel Symfony (Bouton Download). Re démarrer le pc, se rendre dans le terminal à l’endroit où se trouve le projet et on peut maintenant lancer le serveur avec :**

* **server :start**

## Commençons par inspecter notre dossier.

On va s’intéressé particulièrement au dossier **src** qui va contenir l’entièreté des **codes** **php** de notre application ainsi qu’au dossier **template** qui lui va nous permettre de **gérer** nos **fichiers d’affichage**/rendu avec **twig**.

***Ouvrir un 2eme onglet de terminal dans Visuel Studio Code pour ce qui va suivre !***

Pour créer un controller qui va se retrouver au sein du dossier controller.

* **Php bin/console make :controller**

make:controller ⬄ Permet de fabriquer un controller simplement !

Choisir un nom pour la classe controller : 1ere lettre en Maj et le reste en camelCase

* **BlogController**

**Un gros bouton vert avec le mot « Success ! » apparait.**

Ma ligne de commande à créer dans le dossier des controllers un fichier qui s’appelle **BlogController.php** et qui **contient** une **classe** et dans notre dossier templates un dossier qui s’appelle **blog** avec déjà un premier fichier qui s’appelle **index.html.twig** pour qu’on puisse tout de suite commencer à travailler.

# Les Annotations

Inspections ce nouveau fichier BlogController.php

N’importe quel commentaire qui commence par un **@** est une **annotation.**

Ici, **cette annotation s’appelle route**, on explique à Symfony que **quand un navigateur appellera mon site.com/blog , voici la fonction que tu dois appeler. La fonction index,** peut importe ce qu’elle fait.

Ici, notre **but dans cette fonction est de traiter la demande et de renvoyer une réponse. Ici, je renvoie le fait d’afficher un fichier html qui s’appel index.html.twig et qui se trouve dans le dossier blog.** On remarque que on a pas préciser que ce dossier se trouve dans le dossier template car Symfony sait que tout ses templates sont dans ce dossier-là. Donc on ne met que blog/index.html et on comprend que c’est bien **ce fichier twig qui nous permet de rendre un affichage.**

## Première page avec Symfony !

Se rendre sur : <http://localhost:8000/blog>

Je vois que Symfony m’a créé une nouvelle page, elle me dit que si je veux **modifier les traitement** pour cette page ça se trouve **dans BlogController.php**, si je veux **modifier l’affichage** pour cette page ça se trouve **dans blog.index.html.twig**: test : Changer le mot Hello en salut !

## Création de l’ensemble des routes qui serviront à ce blog.

Se rendre dans le BlogController.php et créer la fonction public home. Car je veux que ce soit ça qui soit appeler quand je vais sur mon-site.com/**rien** Donc, ça sera la page d’accueil du site.

# Langage Twig

## Simplicité

-Facilite l’écriture des affichages

-Apporte beaucoup de fonctionnalités (que le templating php avait du mal à nous apporter).

## Absence de PHP

-Permet d’abstraire les affichages de balises PHP

-Plus simple pour un intégrateur

Twig existe en dehors de Symfony !

{# Interpolation : {{ var }} <=> Affiche le contenu d'une variable ! #}

{# Commandes : {% cmd %} <=> Permet d'appeler une commande / structure de contrôle ! / Les commandes c'est tout les traitements possible. #}

# Mise en place : On créé des pages !

## Boostswatch.com ⬄ Des thèmes bootstrap très sympas !

Choisir un thème, par exemple Flatly. Aller sur Flatly et cliquer sur l’onglet Flatly en haut et choisir bootstrap.ss

Dans Twig si on regarde notre dossier templates on a effectivement le dossier blog avec les vues qui concerne le blog mais on a aussi un fichier qui s’appelle base.html.twig qui constitue un template de base. Si je veux que quelque chose se retrouve dupliquer sur toute mes pages, j’ai intérêt à les mettre dans ce template de base puis à chaque fois que je vais créer une autre page plus tard, je dirais qu’elle s’inscrit dans ce template. On remarque que dans ce fichier base on a plusieurs blocks :

block title

block stylesheets

block body

block javascript

Ces blocks sont des emplacements que toutes les autres pages pourront personnaliser. En fonction des pages que je vais créer, je vais demander à chacune d’entre elles de se placer par exemple au sein du block body et il faut imaginer par exemple que chaque page va mettre du contenu dans le block body ou dans le block title, etc. Donc chaque page va pouvoir personnaliser ces différents blocks mais tout ce qui se trouve dans ce template de base se trouvera dupliquer sur toute mes pages.

Donc ce que je vais faire, juste au-dessus du block stylesheets, me créer une balise link et mettre dans le href le lien vers mon css. Désormais, toute mes pages bénéficient de ce lien vers ce css.

Ce que je vais faire aussi, c’est prendre la navbar qui m’intéresse le plus.

Sur chaque composant, il y a 2 petites flèches (<>) sur lesquelles on peut cliquer pour voir son code source. Je copie la navbar que je veux et je vais la coller juste en dessous de mon block body et je supprime le formulaire car il ne me sert à rien pour le moment. Je renomme mon lien <a> et je garde que deux <li>, j’enlève la classe active du 1er <li> et je le fait aller vers Articles et le 2eme sera par exemple créer un article. Je peux aussi modifier les chemins des 2 <li> /blog et blog/new mais on ne les créera pas pour le moment. Pareil pour ma page d’accueil car je sais qu’elle est dans /

Ensuite, je veux que tout mon contenu soit dans une div bootstrap qui aura comme class container  et donc je veux que mon block body soit à l’intérieur d’un container comme ça je sais que chaque page qui enrichira ce blog, le contenu sera contenu.

    <div class="container">

        {% block body %}{% endblock %}

    </div>

### Comment faire en sorte qu’une page s’inscrive dans ce template la ?

Par exemple dans ma page [home.html.twig](http://home.html.twig) je peux utiliser la commande

{% extends 'base.html.twig' %} {# TWIG : EXTENDS <=> Permet de placer un gabarit au sein d'un template général // J'hérite du template-thème de base. #}

Attention, je ne peux pas l’utiliser tel quel. Si j’hérite du template base il faut absolument que mon contenu soit placer dans un certain block. Ici j’ai envie de modifier ce qui se trouve dans le block body donc je vais mettre tout mon code à l’intérieur et je n’oubli pas de mettre la commande {% endblock %} à la fin.

{% block body %} {# TWIG : Surcharge de blocks <=> On projète du contenu dans les différents blocks du template #}