

TP1 : Symfony

1. Installation

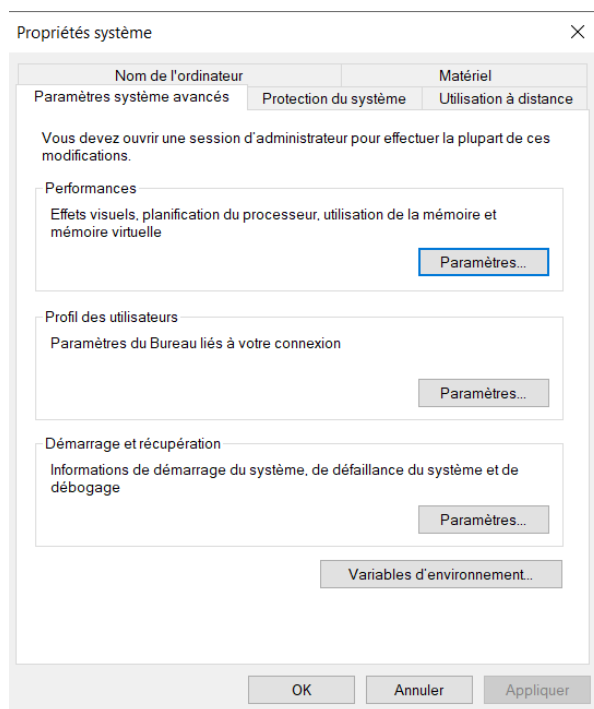
Sur le site de symfony : <https://symfony.com/>

Quelle version minimum de php est requise ?

Installer les outils suivants :

● Variables d'environnement en utilisant wampserver (à télécharger et à installer)

Ajouter une variable :



Dans les variables systèmes :

Variables d'environnement



Variables utilisateur pour admin

Variable	Valeur
OneDrive	C:\Users\admin\OneDrive
Path	C:\Users\admin\AppData\Local\Programs\Python\Python310\...
TEMP	C:\Users\admin\AppData\Local\Temp
TMP	C:\Users\admin\AppData\Local\Temp

Nouvelle...

Modifier...

Supprimer

Variables système

Variable	Valeur
OS	Windows_NT
Path	C:\wamp64\bin\php\php8.2.0;C:\Program Files (x86)\VMware...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
POWERSHELL_DISTRIBUTI...	MSI:Windows 10 Enterprise
PROCESSOR_ARCHITECTU...	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 151 Stepping 5, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	070F

Nouvelle...

Modifier...

Supprimer

Puis : ajouter, parcourir et sélectionner le dossier :

C:\wamp64\bin\php\php8.3.6

C'est la dernière version disponible

Attention : il faudra toujours avoir cette version définie comme variables d'environnement sur les PC pour que votre serveur (et donc votre site) fonctionne !

Vérifier la version de php avec la commande sur le cmd : php -v

● Git : à télécharger <https://git-scm.com/downloads>

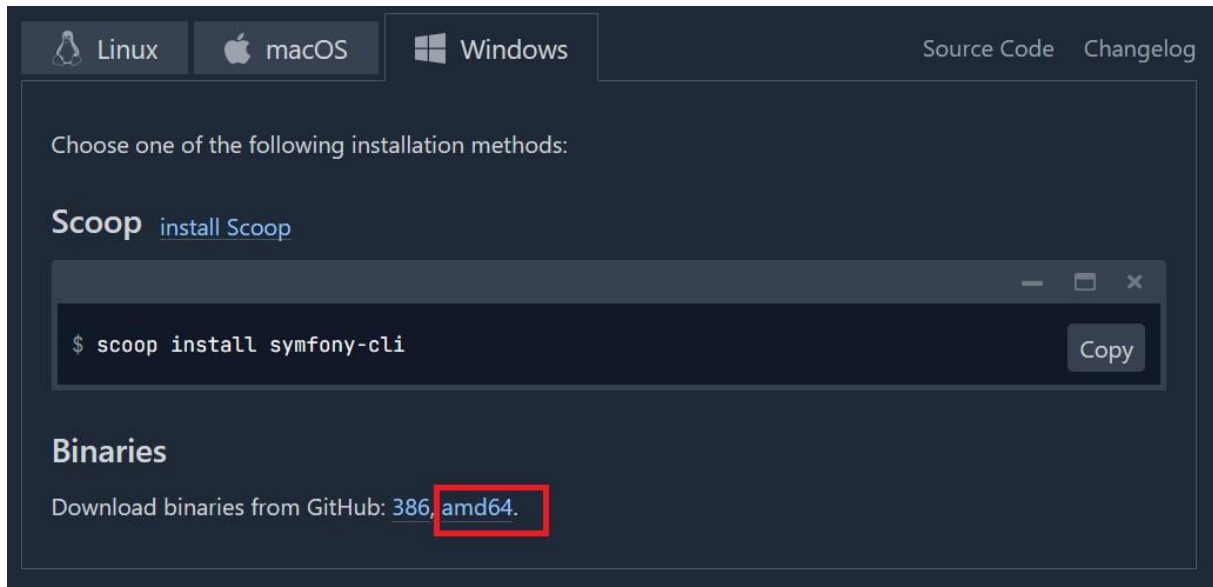
Vérifier avec la commande : git -v

● Composer : à télécharger <https://getcomposer.org/download/>

Vérifier avec la commande : `composer -V`

- Symfony-CLI : téléchargement et / ou ligne de commande

<https://symfony.com/download> et cliquer sur amd64 pour une version x64 :



Ou installez scoop : <https://scoop.sh/>

Pour cela faire les commandes depuis powershell :

```
> Set-ExecutionPolicy RemoteSigned -Scope CurrentUser  
> Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
```

Puis utiliser la commande :

```
$ scoop install symfony-cli
```

Pour vérifier :

```
Symfony -V
```

puis

```
symfony check:requirements
```

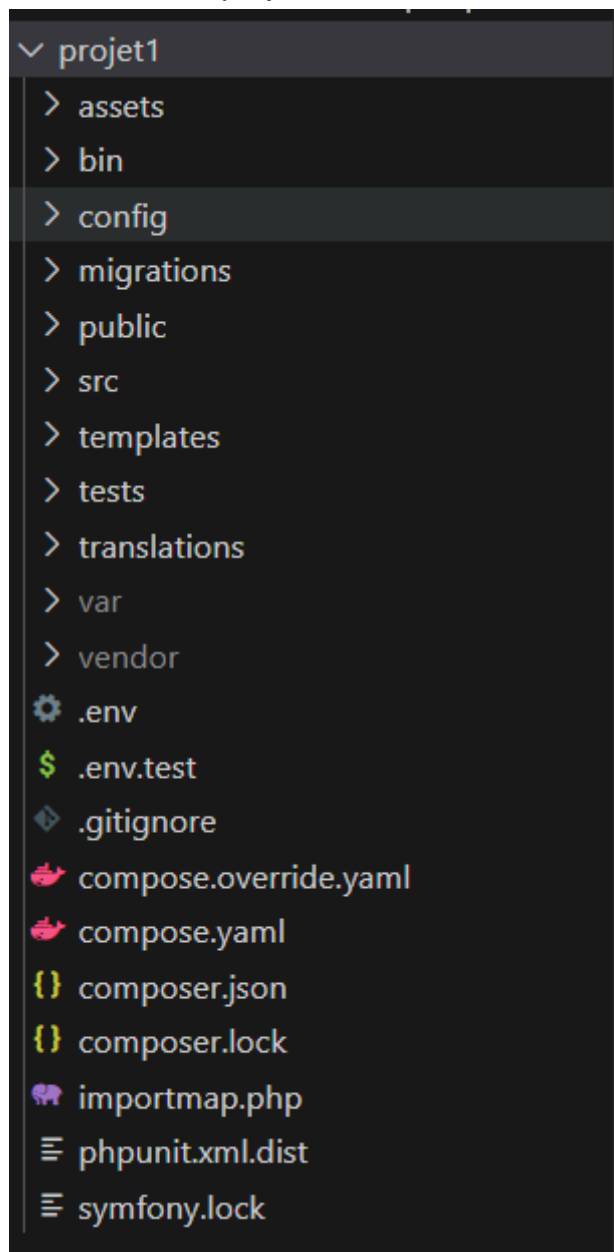
Pour créer un projet symfony, dans un nouveau dossier que vous avez créé, taper la commande :

```
symfony new --webapp NOM_de_votre_Projet
```

Exemple de message que vous devez obtenir :

```
[OK] Your project is now ready in C:\Users\admin\Documents\TPsymfony\projet1
```

Le contenu du projet créé :



Placez-vous, avec une commande sur un terminal, dans le dossier portant le nom du projet que vous venez de créer, exemple :

```
cd projet1
```

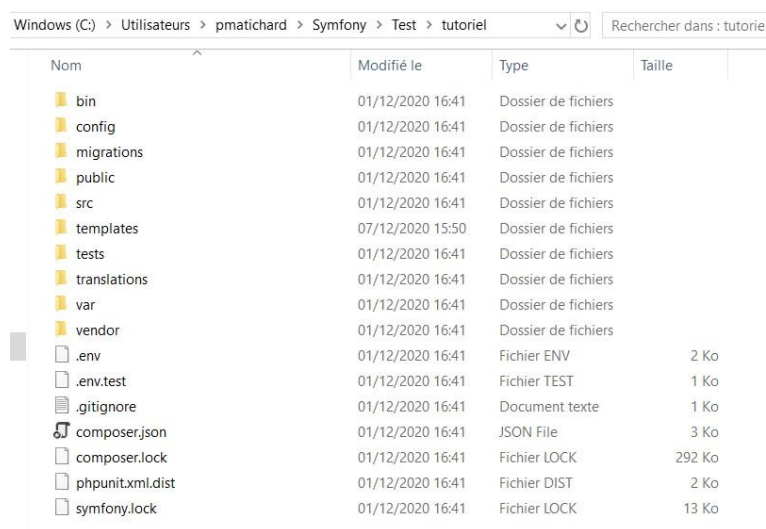
2. Créer une page sur Symfony

a. Explications

Les contrôleurs, en quoi ça consiste :

- 1-Navigateur appelle une route (L'URL taper par l'utilisateur)
- 2-Serveur reçoit la requête http
- 3-Serveur renvoie une réponse http
- 4-Navigateur utilise la réponse (affiche la page Web)

La structure des dossiers d'un projet symfony :



Nom	Modifié le	Type	Taille
bin	01/12/2020 16:41	Dossier de fichiers	
config	01/12/2020 16:41	Dossier de fichiers	
migrations	01/12/2020 16:41	Dossier de fichiers	
public	01/12/2020 16:41	Dossier de fichiers	
src	01/12/2020 16:41	Dossier de fichiers	
templates	07/12/2020 15:50	Dossier de fichiers	
tests	01/12/2020 16:41	Dossier de fichiers	
translations	01/12/2020 16:41	Dossier de fichiers	
var	01/12/2020 16:41	Dossier de fichiers	
vendor	01/12/2020 16:41	Dossier de fichiers	
.env	01/12/2020 16:41	Fichier ENV	2 Ko
.env.test	01/12/2020 16:41	Fichier TEST	1 Ko
.gitignore	01/12/2020 16:41	Document texte	1 Ko
composer.json	01/12/2020 16:41	JSON File	3 Ko
composer.lock	01/12/2020 16:41	Fichier LOCK	292 Ko
phpunit.xml.dist	01/12/2020 16:41	Fichier DIST	2 Ko
symfony.lock	01/12/2020 16:41	Fichier LOCK	13 Ko

Vous travaillerez, dans un premier temps, sur les 2 dossiers suivants :

- Src : code php de notre application
- Templates : fichiers d'affichage (utilise le langage TWIG)

b. Créer un controller

Pour créer un contrôleur (figure au sein du dossier src/Controller) :

Dans l'invite de commandes, en utilisant votre chemin vers le dossier que vous avez créés et hébergeant votre projet symfony (voir l'exemple ci-dessous), taper la commande (encadrée en jaune) :

```
C:\Users\pmatichard\Symfony\Test\tutoriel> php bin/console make:controller
```

Puis, on vous demande de choisir un nom pour votre classe contrôleur :

Choose a name for your controller class (e.g. VictoriousChefController):

Attention à bien respecter la structure de nommage, qui est donné ci-dessus, la première lettre de chaque terme en majuscule, exemple BlogController

>BlogController

Vous venez de créer un fichier « BlogController.php », dans le dossier src/Controller, qui contient une classe (class).

Dans le dossier « templates », un dossier qui se nomme « Blog » a été créé. Il contient un fichier « index.html.twig » qui vous permettra de commencer à travailler sur votre application.

Ouvrir « BlogController.php ». Vous devez voir apparaître :

```
<?php
```

```
namespace App\Controller;
```

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

```
use Symfony\Component\HttpFoundation\Response; use
```

```
Symfony\Component\Routing\Annotation\Route;
```

```
class BlogController extends AbstractController
```

```
{
```

```
    #[Route('/blog', name: 'app_blog')]
```

```
    public function index(): Response
```

```
    {
```

```
        return $this->render('blog/index.html.twig', [  
            'controller_name' => 'BlogController',  
        ]);
```

```
    } }
```

En jaune, nous voyons une classe « BlogController » qui bénéficie d'une autre classe Controller.

Votre « blogController » contient maintenant des lignes de codes, créées par symfony, qui contient de nombreuses fonctionnalités et vous allez en profiter.

En vert, il s'agit d'une annotation. Elle commence avec un #. Cette annotation s'appelle « route ». Quand un navigateur appellera par exemple, « www.monsite.fr/blog » voici la fonction qu'il exécutera :

La méthode index() : **public function index()**

La méthode traite la demande et renvoie le fait d'afficher un fichier html qui s'appelle « index.html.twig » et qui se trouve dans le dossier templates/Blog .

Pas besoin de préciser que le dossier « Blog » se trouve dans le dossier « templates » car symfony sait que tous les fichiers d'affichage se situent dans le dossier « templates ».

Ouvrez le fichier « index.html.twig » :

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

```
{% block title %}Hello BlogController!{% endblock %}
```

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

```
....
```

Remarque : la syntaxe est du twig.

Dans votre navigateur, taper l'URL : 127.0.0.1:8000/Blog

Attention, il faut que votre serveur symfony soit lancé.

La commande est :

symfony serve

ou

synfony server :start

N'utilisez pas la commande symfony server:start -d, elle lance votre serveur sur un daemon (en tâche de fond) et souvent vous avez du mal à gérer.

Vous devez voir cette page :



Si vous voulez modifier les traitements pour cette page, il faut aller dans :

Your controller at [src/Controller/BlogController.php](#)

Si vous voulez modifier l'affichage pour cette page, il faut aller dans :

Your template at [templates/blog/index.html.twig](#)

Allez dans votre fichier « index.html.twig » sur Visual studio code et modifier en remplaçant « Hello » par « Salut » , puis **sauvegarder**.

Remarque : après chaque modification d'un fichier, pour voir le résultat sur votre navigateur, il faut SAUVEGARDER le fichier.

Sur VScode, dans fichier, cocher auto save

ICI

Actualiser votre page sur votre navigateur et vous devez obtenir :



Salut BlogController! 

This friendly message is coming from:

- Your controller at [src/Controller/BlogController.php](#)
- Your template at [templates/blog/index.html.twig](#)

c. Nouvelle page WEB

Vous allez maintenant créer une nouvelle route, donc une nouvelle page et un nouveau traitement.

Dans le fichier « BlogController.php » (ce fichier contient votre contrôleur), créer une public function, que vous nommerez « home() » :

```
Public function home() : response  
{ ...etc....}
```

Cette fonction sera appelée quand on appelle « monsite.com/ ». Elle sera la page d'accueil du site.

Cette fonction va contenir :

```
return $this->render('blog/home.html.twig') ;
```

Vous remplacerez les, entre les guillemets {} de la fonction publique, par le code ci-dessus.

Maintenant vous devez lier cette fonction à une adresse à l'aide d'une annotation. On ajoute un commentaire contenant l'annotation :

```
#[Route('/blog/home', name: 'app_home')]
```

Vous allez créer un fichier « home.html.twig » dans le dossier « templates/blog ».

Ouvrir ce fichier. Mettre un titre (balise h1), « Bienvenue » :

```
<h1>Bienvenue</h1>
```

Visualiser la page sur votre navigateur.

3. Langage Twig

- Simplicité :

- Facilité d'écriture des affichages
- Apporte beaucoup de fonctionnalités

- Absence de PHP au sein de nos fichiers d'affichage :

- Permet d'abstraire les affichages de balise PHP
- Plus simple pour un intégrateur

Dans twig, vous pouvez la possibilité d'afficher le contenu d'une variable.

Par exemple dans les balises <h1> </h1>, je veux afficher le contenu d'une variable « title » à la place de « Bienvenue ».

On note dans le fichier « home.html.twig » :

```
<h1>{{ title }}</h1>
```

Remarque double accolade = interpolation

Il y a un espace entre {{ et title, idem entre title et }}.

Il en sera de même entre {% et une commande et entre une commande et %}.

Utiliser des commandes entre {% %}

Par exemple, ajouter ces lignes sur la page [home.html.twig](#) :

```
{% if age > 18 %}  
    <p> tu es majeur </p>  
{% else %}  
    <p> tu es mineur </p>  
{% endif %}
```

Puis dans notre contrôleur, il faut ajouter un autre paramètre, un tableau avec la liste des variables et leur valeur :

```
#[Route('/blog/home', name: 'app_home')]
public function home() : response
{
    return $this->render('blog/home.html.twig', [
        'title' => "bienvenue",
        'age' => 31
    ]);
}
```

Afficher votre page WEB à l'aide de votre navigateur.