TP Symfony n°1

Prérequis : Installation de votre environnement Symfony

Il vous faut une version de php 7+.

1)Installation de XAMPP

https://www.apachefriends.org/fr/download.html

2) Installation de symfony CLI

https://symfony.com/download
Depuis Binaries (Download Binaries from github : 386, amd64)

Il faut rajouter symfony dans le path (dans vos variables d'environnement)

3) Installation de GIT

https://git-scm.com/downloads

Il faut rajouter Git dans le path (dans vos variables d'environnement)

4) **Installation de composer**

https://getcomposer.org/download/

5) **Installation de la librairie http-foundation**

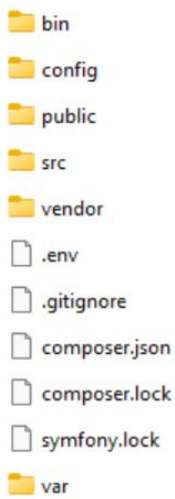
```
composer require symfony/http-foundation
```

Partie 1 : Création de votre projet et lancement de votre application

1) Pour créer un nouveau projet qu'on nommera apptuteur, tapez la commande :

   ```
   symfony new apptuteur
   ```

   Cela va créer l'arborescence suivante du projet :



Ton projet Symfony est dans ce répertoire apptuteur, tu peux créer autant d'applis symfony que tu veux. Chaque application Symfony aura son répertoire dans lequel on pourra lancer le serveur Web.

Petit rappel : Il faut d'abord faire CTRL+D pour sortir du shell …ou CTRL+C pour arrêter le serveur qui tourne dans le terminal…

2) Déplacez vous dans le répertoire

```
cd apptuteur
```

3) Depuis ce répertoire, lancez le serveur propre à Symfony (pas besoin de lancer xampp avant)

```
symfony server:start
```

Une fois, le serveur démarre, il est indiqué sur quelle adresse votre application est disponible.

```
[OK] Web server listening
     The Web server is using PHP CGI 8.2.12
     http://127.0.0.1:8000
```

Partie 2 : A la découverte des routes et des controllers

1. Créez une classe *MainController* dans le répertoire *src/Controller* avec une méthode index avec le code suivant :

```php
<?php
// src/Controller/MainController.php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class MainController
{
    #[Route('/index')]
    public function index(): Response
    {
        return new Response(
            '<html><body>Votre première page</body></html>'
        );
    }
}

?>
```

2. Testez la route suivante http://localhost:8000 . Que constatez-vous ? Quelle est l'URL à taper pour voir l'exécution de la méthode *index* de *MainController* ?

3. Rajoutez une méthode *indexbis* dans votre controlleur, accessible via la route */bonjour*

4. Ajoutez `use Symfony\Component\HttpFoundation\Request` en haut de votre fichier *MainController.php*

Cette librairie va nous permettre d'accéder aux classes `Request` et `Response` (Vous souvenez vous de ce à quoi cela correspond ?)

2. La méthode indexbis fait un affichage du message *"Bonjour X"*, cette méthode va créer un objet `$request` de la façon suivante :

```
$request= Request::createFromGlobals();
```

Puis appelez la méthode query sur cet objet request et la méthode get de la façon suivante :

```
$nom=$request->query->get('nom', 'Inconnu');
```

3.Testez les routes */bonjour* et */bonjour?nom=John*.

A quoi sert `Request::createFromGlobals();` ? A quoi cela équivaut `$request->query->get('nom', 'Inconnu');` si vous aviez utilisé du PHP *"classique"* ?

4. Modifiez votre code pour que à la place de X s'affiche le prénom passé en paramètre.

Au fait, lorsque l'on appelle une URL directement dans le navigateur, de quelle méthode HTTP s'agit-il ?

Partie 3 : Routes paramétrables : valeurs par défaut et contraintes

Une façon plus propre est d'utiliser directement des paramètres dans la route. Par exemple, on souhaitera pouvoir taper la route *bonjour/John* au lieu de *bonjour?nom=John*

1. Modifier l'annotation de votre route et ajouter pour *indexbi*s un paramètre au path *{nom}*

2. Testez à nouveau la route. Que constatez-vous ?

4. Testez la route */bonjour* . Que se passe t il ?

6. Rajoutez dans l'annotation de votre route une option defaults pour le nom qui vaut Inconnu grâce à `defaults: ['nom' => 'Inconnu']`

7. Testez la route */bonjour/35* ? Que constatez vous ?

8. Dans l'annotation, definissez un pré-requis sur le paramêtre avec une expression régulière de la façon suivante `requirements: ['nom' => '[A-Za-z]+']`

9. Retapez la route suivante */bonjour/35* ? Que constatez-vous ?

10. Comment faire pour restreindre l'appel de cette route à un appel en méthode GET ?

11. Créez une méthode indexter toujours dans ce même controleur donc la route sera */calcul/18*

18 étant un paramètre de route, qui sera un chiffre et donc la valeur ne dépassera pas 100.
12. Retirez votre annotation et modifiez le fichier *apptuteur/config/routes.yaml* pour faire en sorte que *indexter* reste accessible, i.e la route est déclarée dans le fichier YAML et non grâce à des annotations.

Partie 4 : Premier template Twig

Twig est un langage de templates qui permet des affichages HTML plus simplement, mais pas seulement ! Nous pourrons produire d'autres formats de HTML

1. Créer un autre controlleur FormController.php avec le code minimal suivant :

```
public function hello($prenom="Bryan", Environment $twig){

return new Response ("Hello $prenom");

}
```

2. Dans le dossier *templates*, créez *hello.html.twig*. Ce fichier possèdera un titre au sens *title* et une balise *h1*

3. Dans *FormController.php*, créez la variable `$html=$twig->render('hello.html.twig');`

Remarque : Pas besoin de préciser à symfony de lui préciser que les templates sont dans le répertoire templates

4. Modifier votre return pour qu'il prenne en paramètre du `Response` cette variable et faites afficher votre page.

5. Modifiez maintenant votre code de *hello.html.twig* et indiquez `{{prenom}}` dans le *h1* En lançant la route, vous verrez que vous obtenez une erreur 500 car la variable n'est pas encore connue Pour cela, on va passer un tableau associatif comme deuxième paramètre de la fonction render `$html=$twig->render('hello.html.twig', ['prenom' => 'Sophie'] );`
6. Testez cette possibilité en remplaçant cette valeur fixe de prénom par le paramètre reçu par votre route.
7. Rajoutez un filtre pour que le prénom s'affiche en majuscule
8. Créez une deuxième méthode *liste()* de route */tuteurs* qui déclare le tableau suivant

```
['tuteurs'=>[ ['nom'=>'Johnson', 'prenom'=>'Paul'], ['nom'=>'Walberg', 'prenom'=>'Mark'] ]]
```

Ce tableau est passé à un template twig que vous créérez pour afficher la liste des tuteurs dans une liste à puces.
9. Créez ensuite un formulaire accessible via la route /search_tuteur, celui-ci affichera un formulaire où l'on peut saisir un nom et il appelera en méthode POST le controlleur *FormController.php* via une méthode *verify* qui declarera le tableau précédent et vérifiera si le tuteur existe ou pas dans les tuteurs présents dans le tableau.
10. Améliorez avec un peu de CSS

Partie 5 : Vérification de vos routes

Testez la commande suivante

```
php bin/console debug:router
```

A quoi sert-elle ?