**Création du projet Symfony**

**Installation du Symfony**

Tu dois dans un premier temps télécharger le binaire de Symfony CLI : <https://symfony.com/download>

Avant, la création de projet Symfony se faisait via Composer. Désormais la création de projet se fait grâce à **Symfony CLI**, un binaire qui fournit un ensemble d’outils pour faciliter la gestion de tes projets Symfony.

Une fois installé, tu devrais pouvoir accéder à la commande « symfony » depuis ta console. Si ce n’est pas le cas, rajoute la en variable d’environnement.

Tu peux tester si cela fonctionne correctement avec la commande :

symfony check:requirements

Maintenant que tu as installé le binaire de Symfony, il est temps de créer le projet

Pour créer ton projet :

symfony new nom\_de\_ton\_projet

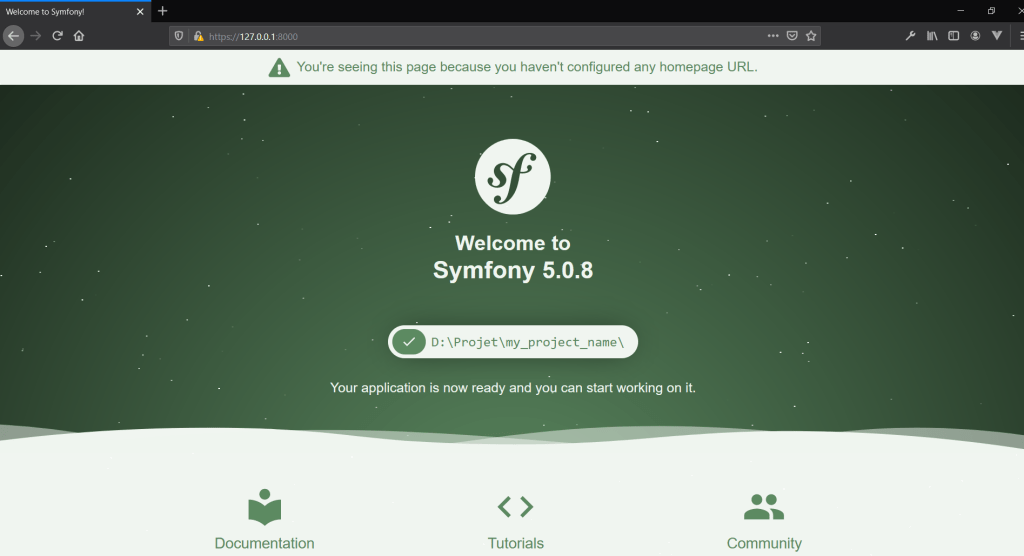
Une fois que la commande est terminée, la base de ton projet est prête mais c’est une coquille vide, il n’y a rien dedans

Vas dans le dossier de ton projet et lances un serveur local :

cd nom\_de\_ton\_projet

symfony server:start

Tu dois pouvoir accéder à l’adresse suivante depuis un navigateur : <https://localhost:8000> et voir apparaître cette page



On va à présent rajouter quelques bundles pour commencer à construire l’application.

**Les logs**

symfony composer req debug --dev

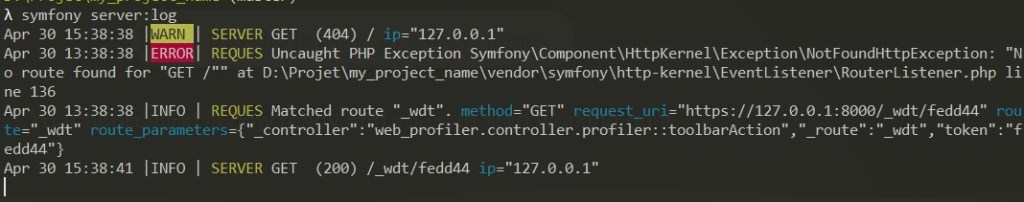
*–dev* permet d’indiquer à composer que ce bundle ne sera installé quand phase de développement. Lorsque ton projet sera en production, ce bundle ne sera pas installé.  
Si tu rafraîchis la page, tu dois voir apparaître une barre en bas :



Je te laisse explorer les différents menus de cette barre. Elle te donnera un nombres d’informations importantes sur ton application (état de la réponse http, durée de chargement des pages, erreurs, …)

Tu peux aussi avoir les logs en temps réel depuis une console :

symfony server:log



**Maker Bundle ou l’outil à tout faire**

Symfny propose le MakerBundle. Une fois installer, tu auras la possibilité de faire de choses incroyables avec : créer des contrôleurs, des entités, des formulaires, …

symfony composer req maker

**Créer un controller**

Tu as surement remarqué que la page d’accueil fonctionne mais retourne une **erreur 404** dans la barre des logs (tout en bas de ton écran)

[Les personnes qui ont lu cet article ont aussi lu :  Comment recevoir un facilement un paiement sur mon site ? (Stripe + Symfony)](https://espritweb.fr/comment-recevoir-un-facilement-un-paiement-sur-mon-site/" \t "_blank)

C’est parce qu’il n’y a pas encore de route définie dans ton projet. Pour cela on va créer un contrôleur.

Les contrôleurs fonctionnent, par défaut, avec le système d’annotation. Il faut donc installer le bundle d’annotation

symfony composer req annotations

C’est parti pour créer notre contrôleur

symfony console make:controller

Donnes un nom à ton contrôleur. Par exemple : DefaultController

Le tour est joué ! Si te rends dans src/Controller, tu verras ton nouveau contrôleur. L’action « index » est créé par défaut. Si tu as utilisé le nom « DefaultController », ta route doit être « /default », tu peux le renommer par « / »

/\*\*

\* @Route("/", name="default")

\*/

public function index()

{

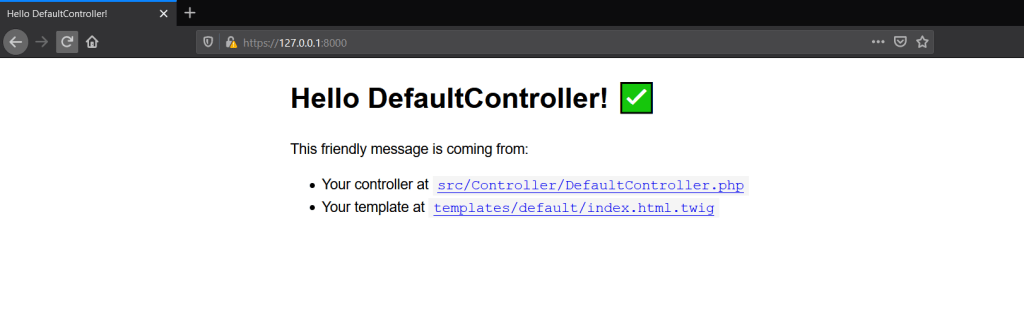
return $this->render('default/index.html.twig', [

'controller\_name' => 'DefaultController',

]);

}

Rafraîchis la page de ton navigateur et tu arrives sur la vue de ton contrôleur.



Jusque-là c’était assez facile, passons aux choses sérieuses avec Webpack.

**Webpack Encore**

Avec l’émergence du JavaScript ces dernières années, de plus en plus de modules, de projets, de librairies reposent sur ce langage. Dans une structure sans Webpack, plus le projet grossit, plus il devient compliqué de maintenir et de s’y retrouver dans tous ce JavaScript si on ne s’organise pas un minimum. Webpack est donc, un moyen plus simple et plus puissant de gérer tous tes JavaScript, tes CSS, tes images de façon organiser. Symfony a intégré Webpack dans ces dernières version. Ce qui est plutôt agréable pour gérer ses projets.

Webpack repose sur **NodeJS** et **Yarn**. Assures toi de les avoir installés :

* <https://nodejs.org/en/download/>
* <https://yarnpkg.com/getting-started/install>

Une fois installé, on va rajouter Webpack à notre application :

symfony composer req webpack

yarn install

*yarn install* permet d’installer tous les modules nécessaires au fonctionnement de Webpack. Si tu es familier avec npm, c’est exactement la même chose. Par la suite, on pourra rajouter des modules avec Yarn. Ex : yarn add bootstrap

Tu as peut-être remarqué qu’un fichier **webpack.config.js** a été créé à la racine du projet. C’est dans ce fichier que se trouve la configuration Webpack.

Jetons-y un coup d’œil :

*.setOutputPath(‘public/build/’)* : définis le chemin où les ressources compilés seront stockées.

Si tu vas dans le dossier public de ton projet, pour le moment il n’y a rien. Exécute cette commande :

yarn encore dev

Un dossier « build » vient de se créer. Dedans on y retrouve les fichiers compilés par Webpack.

*.addEntry(‘app’, ‘./assets/js/app.js’)  
 //.addEntry(‘page1’, ‘./assets/js/page1.js’)*

Ces lignes indiquent à Webpack les différents points d’entrés JavaScript de ton application. Le premier paramètre est **le nom** que nous utiliserons pour intégrer ces points d’entrés dans l’application, le deuxième paramètre est **le chemin** du ficher. Tu peux par exemple créer un fichier JavaScript par page et les ajouter ici.

Un autre fichier a été créé : **assets/js/app.js**

Le fichier ne contient pas grand-chose pour le moment. Il importe simplement un fichier CSS, et fait un console.log d’un texte. C’est avec ce fichier que nous pourrons travailler par la suite.

[Les personnes qui ont lu cet article ont aussi lu :  Comment recevoir un facilement un paiement sur mon site ? (Stripe + Symfony)](https://espritweb.fr/comment-recevoir-un-facilement-un-paiement-sur-mon-site/" \t "_blank)

Cependant si tu te rend dans la console de ton navigateur et que tu rafraîchit la page, tu verras que rien ne se passe, le style de la page est inchangé et le console.log n’a pas fait son travail …

C’est parce que nous n’avons pas encore indiqué à notre application que nous souhaitons utiliser cette entrée (« app »). Faisons le tout de suite.

Dans notre cas, on souhaite que tout ce qui se trouve dans notre fichier **app.js**, soit chargés sur toutes les pages de notre application. Pour se faire, allons dans le fichier **template/base.html.twig**. Ce fichier est la structure HTML de notre application. On va y introduire notre javascript.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>{% block title %}Welcome!{% endblock %}</title>

{% block stylesheets %}

{{ encore\_entry\_link\_tags('app') }}

{% endblock %}

</head>

<body>

{% block body %}{% endblock %}

{% block javascripts %}

{{ encore\_entry\_script\_tags('app') }}

{% endblock %}

</body>

</html>

Grace aux balises **{{ encore\_entry\_link\_tags(‘app’) }}** et **{{ encore\_entry\_script\_tags(‘app’) }}**, twig inclut les bon fichiers Javascript et CSS. Le nom « app » est le nom qui se trouve dans le fichier **webpack.config.js**

Si tu rafraîchis ta page, tu verras que le CSS (**assets/css/app.css**) s’est appliqué, et que le console.log s’est exécuté.

Voilà tu as intégré Webpack à ton application Symfony. Un petit exemple d’utilisation avec **Bootstrap** ça te tente ?

**Ajouter Bootstrap**

Bootstrap est un framework CSS qui permet de pas trop s’embêter avec le CSS quand on ne maîtrise pas trop ça (ce qui est mon cas).

Commençons par ajouter Bootstrap grâce à Yarn

yarn add bootstrap

Tu peux retrouver l’ensemble des modules téléchargés dans le dossier **/node\_modules**

Pour fonctionner, Bootstrap a besoin des librairies jQuery et popper.js, ajoutons-les

yarn add jquery popper.js

Une fois installer, créer un fichier **global.scss** dans « assets/css ». C’est dans ce fichier que l’on vas importer le CSS de Bootstrap.

@import "~bootstrap/scss/bootstrap";

allons dans le fichier **app.js**

//On importe notre fichier global.scss

import './../css/global.scss'

//On ajoute jquery et le Javascript de Bootstrap

const $ = require('jquery');

require('bootstrap');

Il faut ensuite indiquer à webpack que l’on souhaite utiliser du SCSS. Pour cela dé-commenter la ligne ‘*.enableSassLoader*()’

// enables Sass/SCSS support

.enableSassLoader()

Il faut aussi ajouter les modules **sass-loader** et **node-sass**

yarn add sass-loader@^8.0.0 node-sass --dev

Il ne vous reste plus qu’à compiler les fichiers avec Yarn

yarn encore dev

Si tu ne souhaites pas t’embêtez à exécuter cette commande à chaque fois que tu fais une modification dans un fichier, tu peux utiliser la commande suivante :

yarn encore dev –watch

Elle surveillera automatiquement les changements et effectuera un build à chaque changement. Plus besoin de lancer la commande à chaque fois. Tu peux rafraîchir ta page. Bootstrap est maintenant fonctionnel.

Voila, nous venons d’intégrer Webpack à Symfony. Tu peux maintenant rajouter n’importe quel module via Yarn et l’intégrer dans ton projet.