**Installation**

Pour utiliser Symfony il faut installer plusieurs composants en respectant l’ordre suivant :

Composer

Télécharger le fichier d’installation sur le site <https://getcomposer.org/download/>

Executer le fichier.

Scoop

Dans le Powershell, executer la commande suivante :

Invoke-Expression (New-Object System.Net.WebClient).DownloadString('https://get.scoop.sh')

Symfony

Dans le bash, executer :

scoop install symfony-cli

Doctrine

Dans le Bash, executer :

composer require doctrine/doctrine-migrations-bundle

Il faut ensuite synchroniser Doctrine et Symfony

php bin/console doctrine:migrations:sync-metadata-storage

**Création d’un nouveau projet**

Pour une installation standard

symfony new --webapp (ou --full pour les versions plus anciennes) **my\_project**

Pour une installation lts (Long Time Support)

symphony new --full **projet-lts** --version=lts

Pour installer une connexion sécurisé (https) éxecuter en plus :

symfony server:ca:install

**Configuration de l’application**

* Il faut vérifier si tous les prérequis sont installé et activés. Pour cela executer :

symfony check:requirement

- Pour activer des modules il faut les décommenter en enlevant le ; dans le fichier php.ini

- Pour l'accélérateur (opcache) : ajouter la ligne : Zend\_extension="C:\xampp\php\ext\php\_opcache.dll" sous ;zend\_extension=opcache **et** décommenter opcache.enable=1

* Ré-enregistrer le fichier .env en .env.local (A la racine de l’application)
* Pour l'utilisation de Mysql: commenter la ligne de postgresql et décommenter la ligne de Mysql, puis Changer l'utilisateur et le mot de passe (db\_user:db\_password). Ici **root**.
* Remplacer db\_name par le nom de notre base de données. Ici **ecommerce\_sf6**
* Indiquer la version de serveur de Mysql. Ici **10.4.18**

Pour information

* Lors du passage en production, dans le fichier .env.local, il faudra passer l’attribut **APP\_ENV** de **dev** à **prod**.
* **APP\_SECRET** : clé qui génère les jetons CRSF utilisé pour la validation des formulaire. Devra être modifié lors du passage en production.
* **MESSENGER\_TRANSPORT\_DSN** : sert à la gestion de la fil d’attente des mails.
* **MAILER\_DSN** : a décommenter et configurer si on veux intercepter les mails envoyés.

**Création et peuplement de la base de données**

Création de la BDD

symfony console doctrine:database:create

Création de la table utilisateurs

symfony console make:user

**Fichier généré :** dans src/Entity → **nomdufichier**.php

Création ou modification des entités

symfony console make:entity

**Fichier généré :** dans src/Entity → **nomdelatable**.php pour chacune des entités.

Création du fichier avec les requêtes SQL

symfony console make:migration

**Fichier généré :** dans migrations → Version**xxxxxxxxx**.php

Migration des informatioons vers la BDD

symfony console doctrine:migration:migrate

Important

* Pour la creation des clé étrangère il faut, lors de la création de la propriété, définir le type à relation.
* Dans les fichiers php générés, il faut manuellement précisé les propriétés unique en ajoutant : **unique: true**.
* Pour les dates, il faut ajouter : **options: ['default' => 'CURRENT\_TIMESTAMP']** pour que ce soit la date courante qui soit prise en compte.

**Création et gestion des templates**

**Avant de commencer.** Il faut désintaller le composant Webpack de symfony. Ce dernier peut entrainer des erreurs par la suite.

composer remove webpack

* Une fois désinstallé, supprimer le dossier **assets**.
* Dans le fichier **base.html.twig** : vider les blocks stylesheet et javascript dont les contenus sont en fait des reliquats de Webpack.

Création du contrôleur principal

symfony console make:controller **MainController**

**Fichier généré :** dans src/Controller → **MainController**.php

Pour informations

* Il est recommandé de séparer les fichiers communs en plusieurs partie telles que le footer, la nav, etc…., et de les placer dans un sous-dossier de template. Par exemple \_partials. Dans ce cas il faudra utiliser la méthode **include** pour les importer dans un fichiers.
* La méthode **asset()** va chercher les fichier directement dans le dossier **public**.
* Si dans le fichier **index.html.twig** on intègre un block déjà existant dans le fichier **base.html.twig**, il remplacera ce dernier.

En cas de besoin de fichiers css et js personnels

* Créer un dossier **assets** dans **public**, puis les sous-dossiers **css** et **js**. Y placer les fichiers **styles.css** et **script.js**.
  + Les fichiers devront être importés dans **base.html.twig** via la méthode **asset()**
    - <link rel="stylesheet" href="{{ asset('assets/css/styles.css') }}">
    - <script src="{{ asset('assets/js/scripts.js')}}" **defer**></script>
  + L’importation devra se faire en dessous des blocks correspondant afin de permettre la surcharge.
  + De plus, il faut ajouter manuellement l’attribut **defer** afin que le javascript ne démarre pas avant le chargement du DOM (les balises se trouvent dans le <head>).

En cas d’utilisation de Bootstrap

* Télécharger les fichiers bundle de Bootstrap et copier les fichiers suivants :
  + Dans **css** : **bootstrap.min.css** et **bootstrap.min.css.map**.
  + Dans **js** : **bootsrap.bundle.min.js** et **bootstrap.bundle.min.js.map**.
* Importer les fichiers de bootstrap dans base.html.twig en les plaçant au dessus des blocks correspondant.
  + - <link rel="stylesheet" href="{{ asset('assets/css/bootstrap.min.css') }}">
    - <script src="{{ asset('assets/js/bootstrap.bundle.min.js')}}" **defer**></script>
  + Penser à ajouter l’attribut **defer**. Cf plus haut.