##
INSTALLER SCOOP
##

Windows + R => Powershell
=> Terminal Powershell (et pas un autre)

Set-ExecutionPolicy RemoteSigned -scope CurrentUser
=> Répondre "O" pour OUI

iwr -useb get.scoop.sh | iex
=> l'installation peut prendre quelques temps
=> Une fois terminée, on peut vérifier à la racine de l'utilisateur la présence du répertoire "scoop"

A QUOI CA SERT ?

Scoop est un gestionnaire de paquets pour Windows proposant les fonctionnalités suivantes :

Élimine les fenêtres contextuelles d'autorisation
Masque les installateurs et setup (installation silencieuse)
Empêche la pollution de la variable PATH
Évite les effets secondaires inattendus de l'installation et de la désinstallation des programmes
Trouve et installe automatiquement les dépendances
Effectue toutes les étapes de configuration supplémentaires elle-même pour obtenir un programme de travail

Scoop télécharges et gère les paquets de manière portable, les gardant soigneusement isolés dans ~\Scoop.
Il n'installe pas des fichiers à l'extérieur de ce dossier, de plus, vous pouvez placer une installation de scoop où vous le souhaitez.
Ainsi, par défaut, les paquets s'installent dans ~/scoop, soit donc pour Windows le dossier C:\Users\<user>\scoop

##
INSTALLER SYMFONY CLI (Command Line Interface)
##

scoop install symfony-cli
=> puis symfony -V pour vérifier l'installation

A QUOI CA SERT ?

Symfony CLI est un outil polyvalent qui simplifie le développement, la gestion et le déploiement d'applications web Symfony.
Il est essentiel pour les développeurs web utilisant ce framework et peut grandement contribuer à améliorer votre efficacité en tant que développeur web Symfony.

Il a plusieurs utilisations essentielles pour les développeurs web, notamment :
Création de projets Symfony
=> Vous pouvez utiliser Symfony CLI pour créer rapidement un nouveau projet Symfony en utilisant la commande symfony new
Génération de code
=> commandes pour générer automatiquement du code, telles que des contrôleurs, des entités, des formulaires, des migrations de base de données, etc.
Serveur de développement
=> lancer un serveur web de développement en utilisant la commande symfony serve. Cela vous permet de tester votre application localement pendant le développement.
Accès aux outils Symfony
=> donne accès à divers outils et commandes utiles pour la gestion de votre application Symfony.
Par exemple, vous pouvez exécuter des commandes Doctrine pour interagir avec la base de données, des commandes de débogage pour inspecter votre code, etc.
Gestion des dépendances
=> facilite la gestion des dépendances de votre projet en utilisant Composer.

Vous pouvez exécuter des commandes telles que composer install ou composer update pour gérer les packages nécessaires à votre application.
Tests et débogage
=> exécuter des tests unitaires et fonctionnels à l'aide de Symfony CLI en utilisant PHPUnit et les outils de test Symfony intégrés.
Déploiement
=> aider à déployer votre application sur différents environnements, en utilisant des commandes telles que symfony deploy pour faciliter le processus de déploiement.

##
SYMFONY SETUP
##

symfony new -webapp "Nom de l'Application"

cd "Nom de l'Application"

symfony serve
=> tester dans le navigateur http://localhost:8000

Editer le fichier .env pour déclarer le gestionnaire de BDD et la BDD utilisée.
=>
DATABASE_URL="mysql://root:@127.0.0.1:3306/"Nom_Base_de_Donnée"?serverVersion=mariadb-10.4.7&charset=utf8mb4"

php bin/console doctrine:database:create
=> Correspond à "Nom_Base_de_Donnée" mis précedemment dans le DATABASE_URL

php bin/console make:entity Test
=> Créer une entitité "Test" qui demandera des attributs
   Exemple : - Nom
             - string
             - 255
             - NOT NULL

php bin/console make:migration
=> Créer les fichiers de migration

php bin/console doctrine:migrations:migrate
=> Execute la création des tables

composer require orm-fixtures --dev
=> Missing package : to use the make:fixtures command, run : composer require orm-fixtures --dev

php bin/console make:fixtures TestFixtures
=> Créer un fichier fixtures puis le modifier comme suit :
   Exemple : new Test();
             $test->setNom('Step by Step Symfony')
             ->setDescription('Processus de création d'un projet Symfony');
             $manager->persist($test);

=> dans "Test"Fixtures.html ajouter use App\Entity\"Test";

php bin/console doctrine:fixtures:load
=> Charge les fixtures
=> database "test" will be purged. Do you want to continue? (yes/no) [no]:
=> Ecrire "yes" et appuyer sur entrée

php bin/console make:controller TestController
=> Créer les routes
   Exemple :

```
    #[Route('/test', name: 'app_test')]
    public function index(TestRepository $ar): Response
    {
        return $this->render('test/index.html.twig', [
            'test' => $ar->findAll(),
            'controller_name' => 'TestController',
            'route_name' => '/corps'
        ]);
    }
```

=> Après avoir créer le Controller, il se peut que l'affichage ne fonctionne pas et renvoi une erreur sur :
localhost:8000/test
=> Si c'est le cas alors :
- symfony server:stop
- symfony server:start --no-tls (ou symfony serve) "--no-tls" empêche l'affichage du warning dans le terminal

La page devrait s'afficher correctement.