

Initiation à Symfony 5.4

Projet « Liste de Courses »

- ✓ Travail **INDIVIDUEL** à réaliser sur une séance de **4 heures**,
- ✓ **Pas de Compte-rendu de TP** à rendre,
- ✓ Réalisation à présenter à l'enseignant en fin de séance.

Objectifs

- Réaliser une application de création de listes de courses,
- Découvrir le *Framework Symfony* dans sa version 5.4.

Ressources

- Toutes les étapes de réalisation sont détaillées dans ce TP. Vous n'aurez besoin que d'un PC de développement et d'une connexion Internet.
- *Wampserver* doit être installé et fonctionnel sur votre poste (idéalement avec *PHP 8.1.**).

On demande

- ✓ Soyez attentif, ne brulez pas les étapes. Validez chaque point avant de passer au suivant.

L'objectif de ce TP sera de réaliser un outil qui vous aidera à faire vos courses.

De chez vous, depuis l'ordinateur, vous pourrez préparer votre liste de courses. Ensuite, dans le magasin, vous pourrez cocher les articles mis dans votre caddy avec votre smartphone¹.



¹ Ce premier TP s'arrête à la création des produits.

Table des matières

Avant de commencer : Préparer son poste de travail !	4
Installation des outils	4
Installation de Composer	4
Installation de Scoop	6
Installation de Git	6
Installation de Symfony CLI	8
Quelques réglages	8
Ajustement de la configuration PHP	8
Vérification des exigences Symfony	9
Création du projet Symfony « <i>Mes_Courses</i> »	10
Création d'un projet « <i>Application Web</i> »	10
Création d'un certificat	10
Démarrage du serveur Symfony	11
Création d'un projet <i>Php</i> sous <i>NetBeans</i>	12
Travailler avec GitHub	13
Créer un dépôt local	13
Publier votre dépôt	13
Le M.V.C. (Modèle - Vue - Contrôleur)	14
Introduction au MVC	14
Organisation des fichiers de votre projet	14
Création d'une première page	15
Le contrôleur « <i>MainController</i> »	15
Création d'une action (méthode) « <i>index</i> »	15
Le template TWIG	16
Résultat de notre première page Home	17
Intégration de <i>Bootstrap</i>	17
La version simple : Partir du template Bootstrap	17
Utilisation des icônes Bootstrap	18
La procédure conseillée : Utilisation du Webpack Encore	19
Installation de la base de données	19
Avant de commencer	19
Création de la base de données sous phpMyAdmin	19
Configuration du fichier <i>.env.local</i>	20
Aperçu du MCD du projet	21

Les entités	21
L'entité : <i>Zone</i>	21
Qu'avons-nous besoin de renseigner ?	21
Création de l'entité	21
Migration vers la base de données	22
Le fichier <i>Zone.php</i>	23
Les fixtures	23
Page d'affichage des Zones	24
Le contrôleur <i>ZoneController</i>	24
Création du contrôleur	24
L'action <i>index()</i>	25
Template <i>zone/index.twig.php</i>	26
Mise en forme avec Bootstrap	27
Création d'une nouvelle Zone	28
Génération d'un formulaire	28
Adaptation du contrôleur	28
Ajout des dépendances	28
Ajout de l'action <i>new</i>	29
Création de la vue	29
Et avec Bootstrap ?	30
Conclusion de cette première partie	31
Template de base : Ajout d'une Navbar et redéfinition des blocks	32
Lien vers la page « Nouvelle Zone »	32
L'entité <i>Produit</i>	33
Création de l'entité	33
Migration de la BDD	34
Gagner du temps avec le CRUD !	35
Adaptation du CRUD	36
La page <i>index.html.twig</i>	36
Le formulaire <i>ProduitType</i>	37
Les pages <i>new.html.twig</i> et <i>edit.html.twig</i>	38
Affichage de la liste des produits	38
Rendu final	38
Tri de la liste des produits par zone	40

Avant de commencer : Préparer son poste de travail !

Installation des outils

Pour commencer, il est nécessaire de s'assurer d'avoir une configuration logicielle conforme aux besoins d'un projet Symfony.

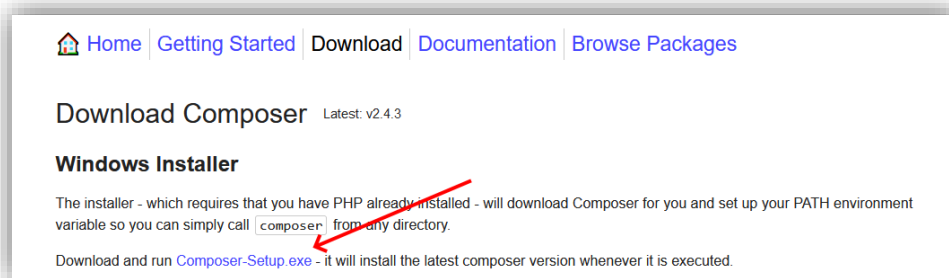
Nous utiliserons les outils suivants :

- ✓ *Composer*
- ✓ *Scoop*
- ✓ *Git*
- ✓ *Symfony CLI*

Partons du principe qu'une version récente de *Wampserver* est installée sur vos machines avec un PHP8.1.x.

Installation de Composer

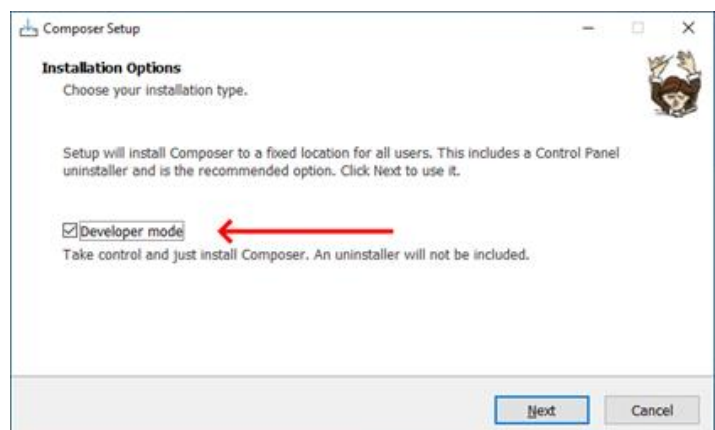
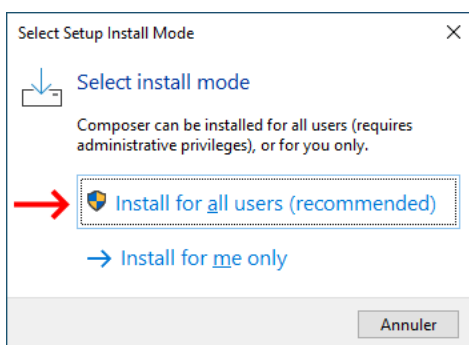
Téléchargez Composer à l'adresse suivante et installez-le : <https://getcomposer.org/Composer-Setup.exe>



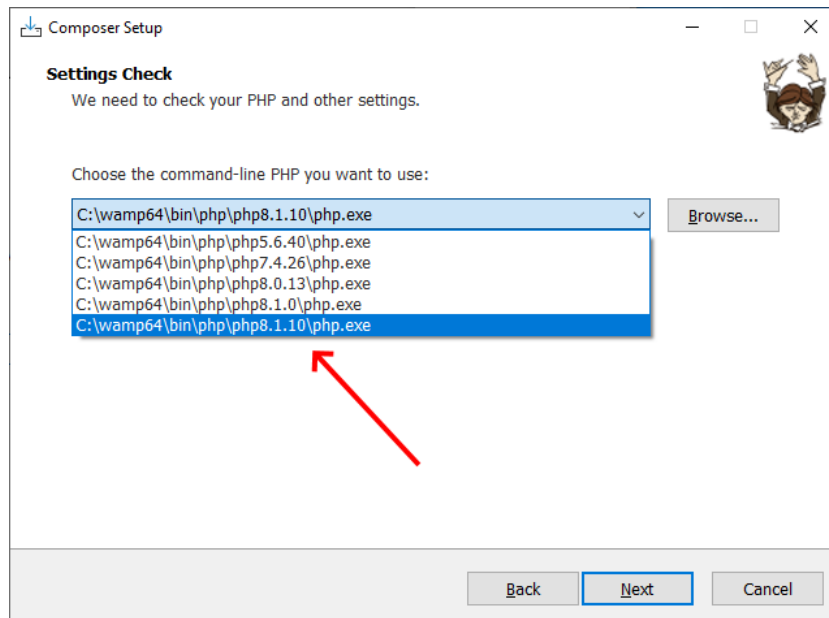
Attention ! Le programme est très long (voir bloquant) à se lancer. Il faut être patient et attendre, l'installation finit par commencer au bout de quelques dizaines de secondes.

Installez composer pour tous les utilisateurs de votre poste.

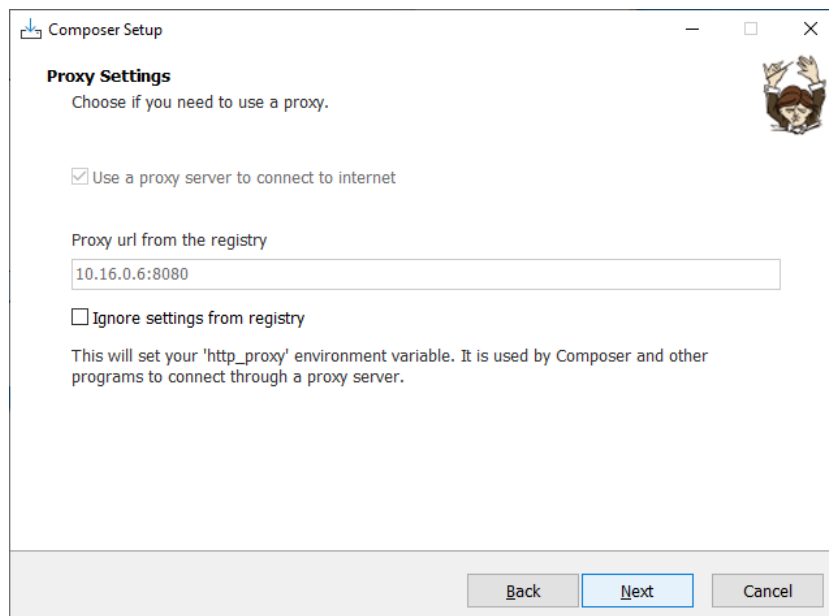
Choisissez l'option « *Developer mode* » et validez une à une les étapes suivantes.



Quelques fenêtres plus loin, **sélectionnez la version de PHP la plus récente** (dans notre cas, PHP8.1.10 installée avec Wampserver).



Validez l'utilisation du proxy avec les paramètres habituels.

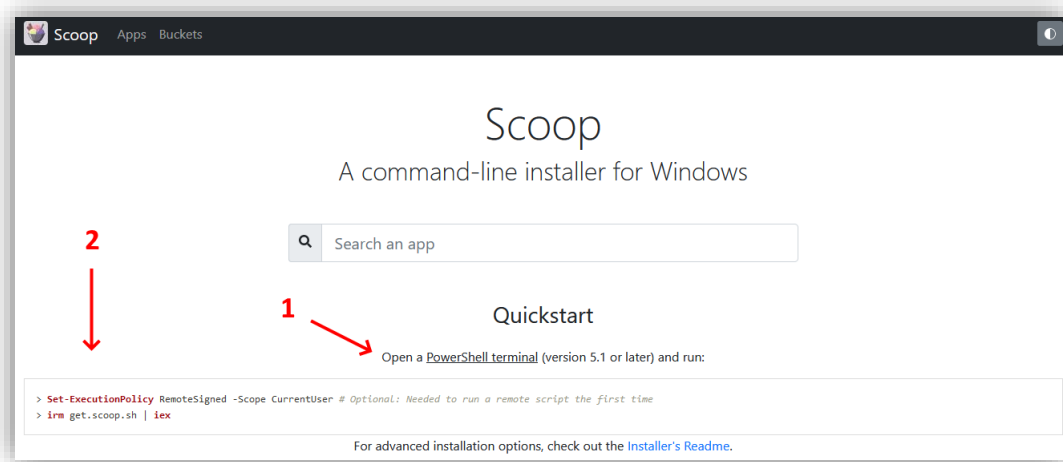


Finalisez l'installation en cliquant sur suivant.

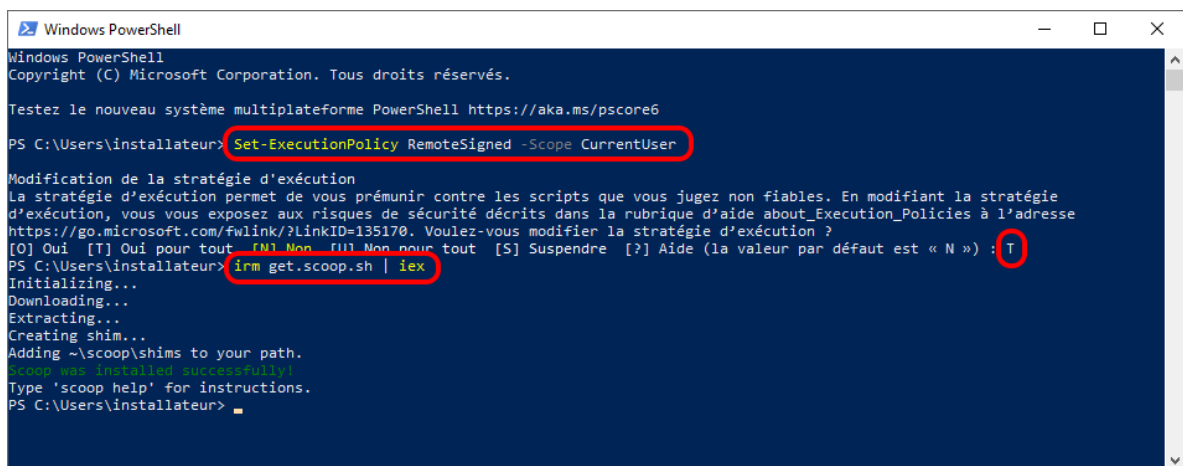
Installation de Scoop

Rendez-vous à l'adresse : <https://scoop.sh/>

Ouvrez une fenêtre PowerShell et recopiez les 2 lignes indiquées sur la page d'accueil.



Autorisez le script l'installation de tous les scripts en appuyant sur 'T'.

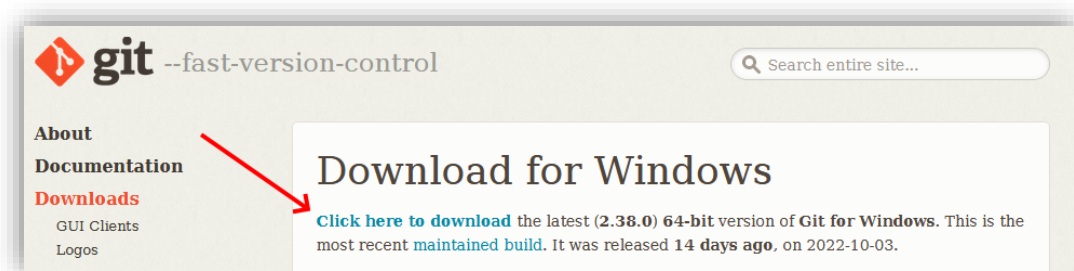


Scoop vous permettra d'installer l'outil *Symfony CLI*.

Installation de Git

Symfony utilise les commandes Git pour gérer les fichiers à inclure ou exclure dans les commit de vos projets.

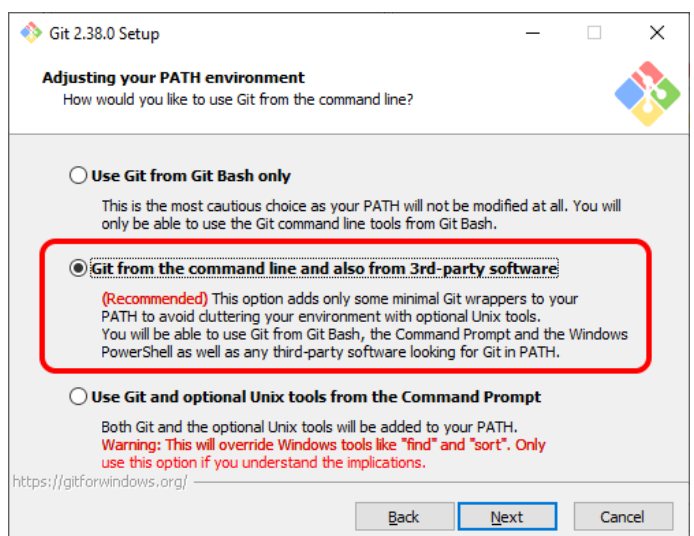
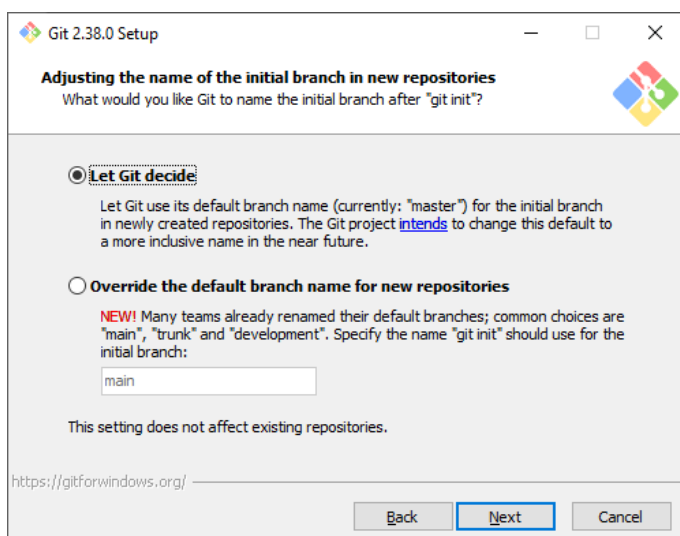
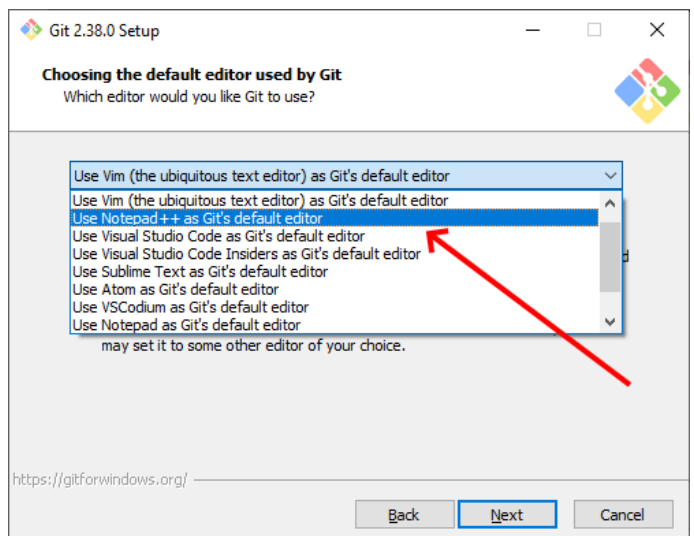
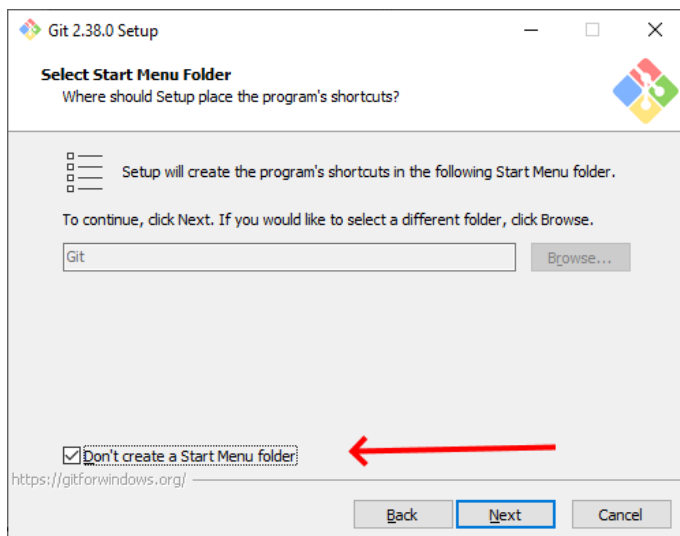
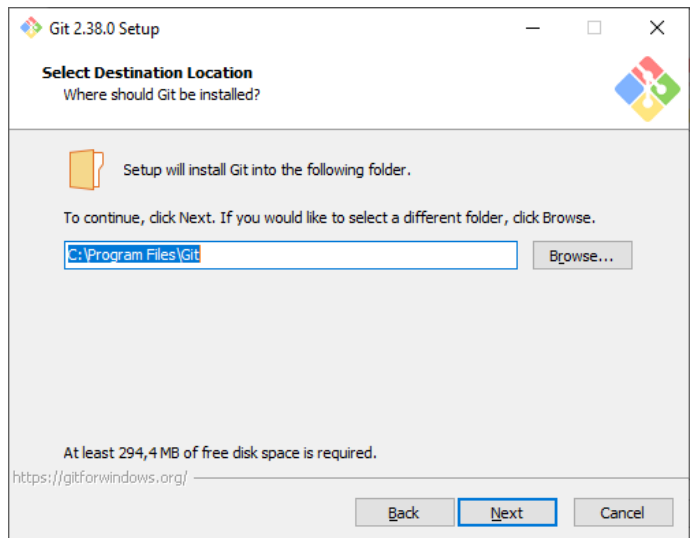
Téléchargez le programme d'installation pour Windows à l'adresse suivante : <https://git-scm.com/download/win>

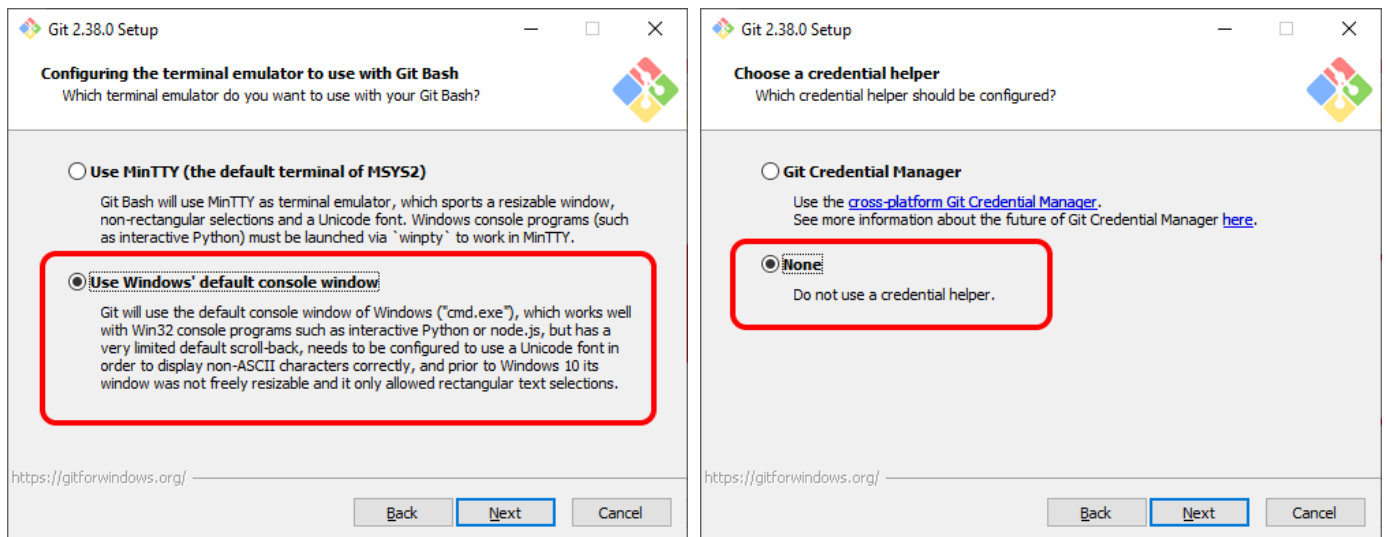




Attention ! Tout comme composer, il y a des longueurs au début de l'installation. Soyez patients.

Suivez un par un les écrans ci-dessous jusqu'à la fin de l'installation en sélectionnant les mêmes options.





Git est maintenant installé !

Installation de Symfony CLI

L'outil *Symfony CLI* vous permettra entre autres de créer des projets et démarrer le serveur que chacun d'eux intègre.

Rendez-vous sur la documentation à l'adresse suivante : <https://symfony.com/download>

Utilisez la commande suivante dans la console CMD pour l'installer : `scoop install symfony-cli`

```

Invite de commandes
Microsoft Windows [version 10.0.19044.1288]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\installateur> scoop install symfony-cli
Installing 'symfony-cli' (5.4.16) [64bit] from main bucket
symfony-cli_windows_amd64.zip (5,2 MB) [=====] 100%
Checking hash of symfony-cli_windows_amd64.zip ... ok.
Extracting symfony-cli_windows_amd64.zip ... done.
Linking ~\scoop\apps\symfony-cli\current => ~\scoop\apps\symfony-cli\5.4.16
Creating shim for 'symfony'.
'symfony-cli' (5.4.16) was installed successfully!

C:\Users\installateur>

```

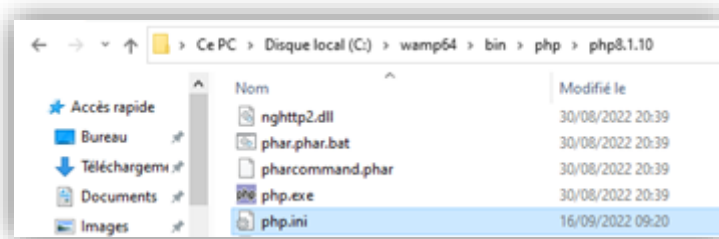
Votre PC est prêt !

Quelques réglages

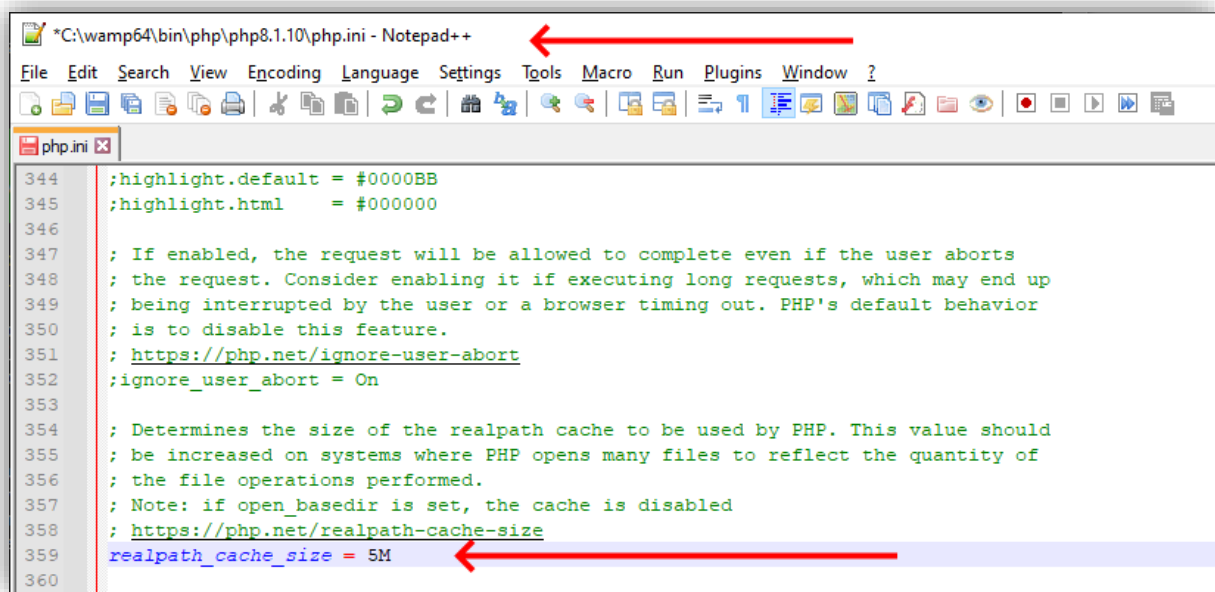
Ajustement de la configuration PHP

Pour des performances optimales, *Symfony* peut vous conseiller d'ajuster quelques réglages.

Editez le `php.ini` qui se trouve à l'adresse : `C:\wamp64\bin\php\php8.1.10`



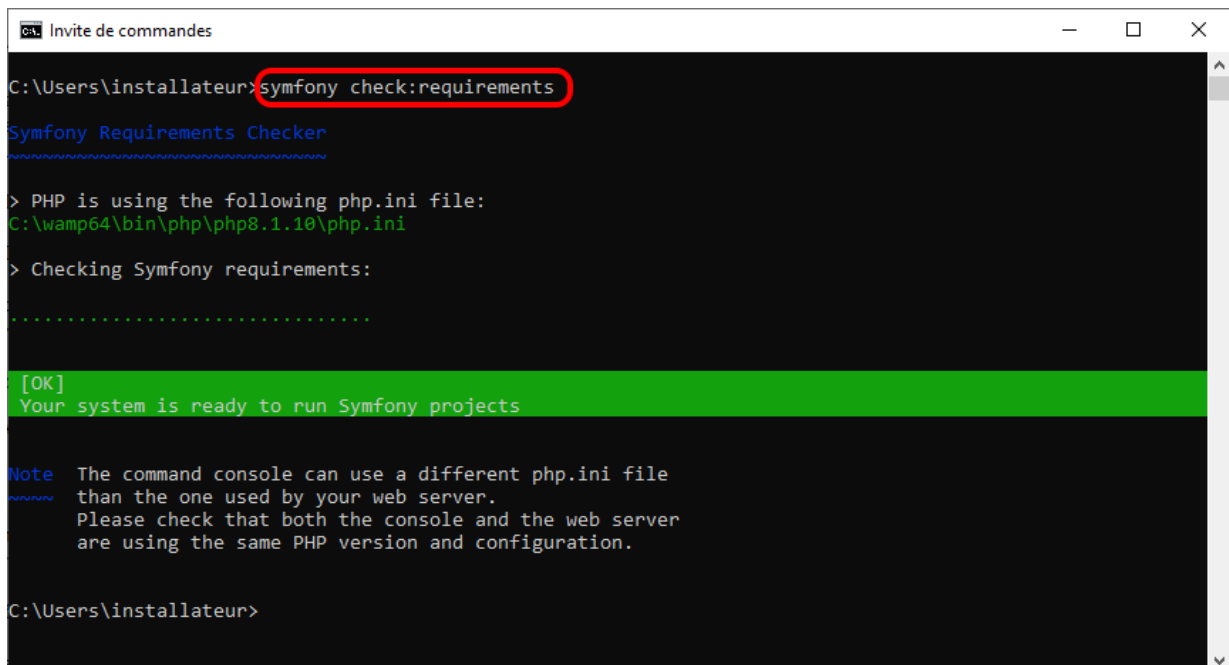
Passez le "realpath_cache_size" à 5M et décommentez la ligne (enlevez le ';' à gauche)



Faites un clic-droit sur l'icône de Wampserver, puis **Redémarrez les services** pour prendre en considération les modifications (n'oubliez pas de sauvegarder le fichier *php.ini*).

Vérification des exigences Symfony

Dans la console, tapez la commande : `symfony check:requirements`



Votre poste est prêt pour un premier projet.

Création du projet Symfony « Mes_Courses »

Nous allons, à travers ce TP, réaliser un projet simple qui nous permettra de créer une liste de courses en version Web. L'objectif sera de comprendre les bases de Symfony et le principe du modèle MVC.

Création d'un projet « Application Web »

Tout d'abord, créez un répertoire « GitHub » dans mes documents pour ranger vos projets Symfony :

C:\Users\prenom.nom.SN\Documents\GitHub

Dans documents (via console), tapez la commande : `symfony new mes_courses --version=5.4 --webapp`

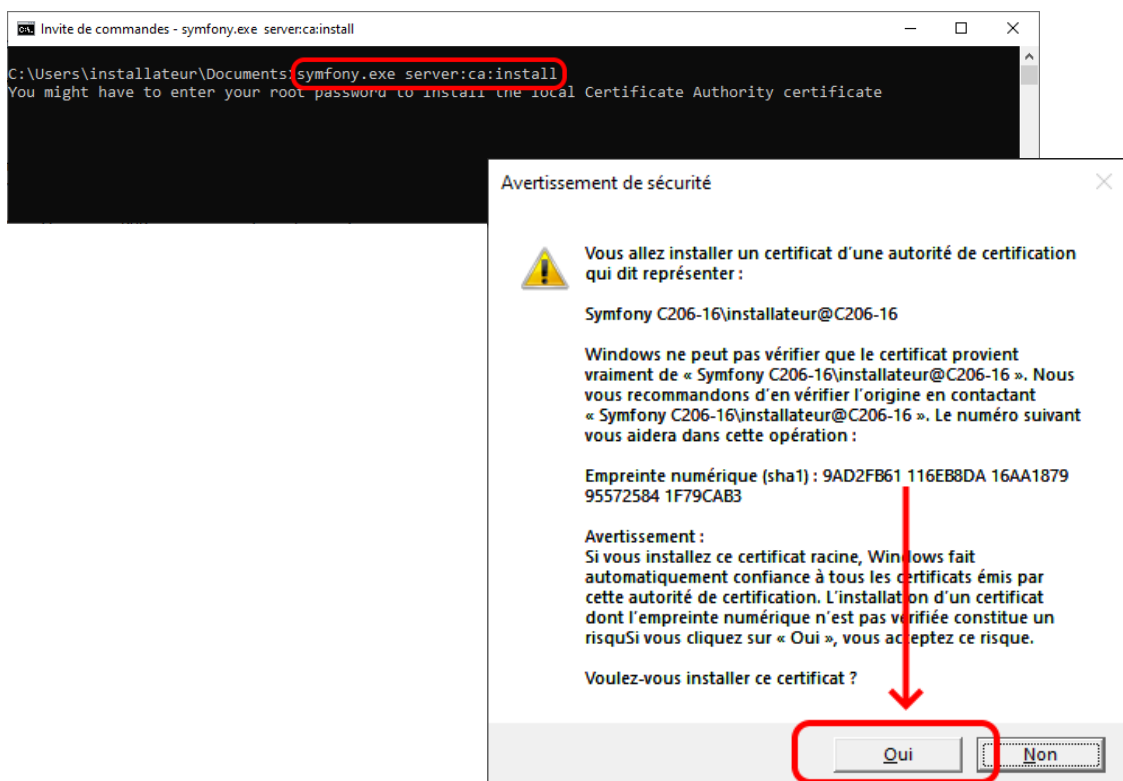


Votre premier projet Symfony est installé !

Création d'un certificat

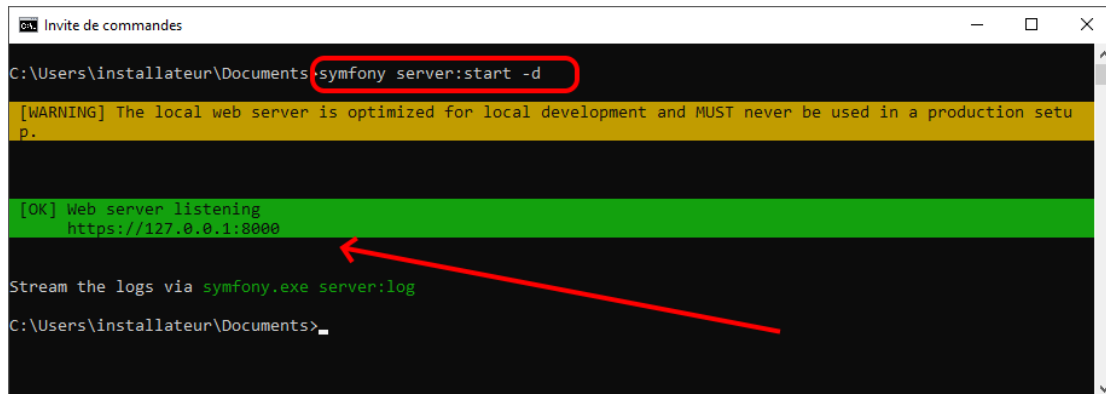
Pour utiliser le protocole HTTPS, votre projet a besoin d'un certificat. Symfony propose une commande qui vous permettra d'en installer un localement.

Dans la console, tapez la commande : `symfony.exe server:ca:install`



Démarrage du serveur Symfony

Dans la console, tapez la commande suivante pour démarrer le serveur : `symfony server:start -d`



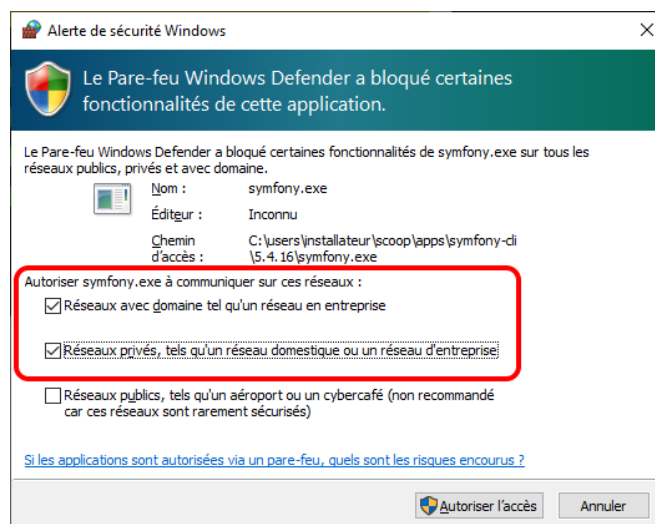
```
CA Invite de commandes
C:\Users\installateur\Documents>symfony server:start -d

[WARNING] The local web server is optimized for local development and MUST never be used in a production setup.

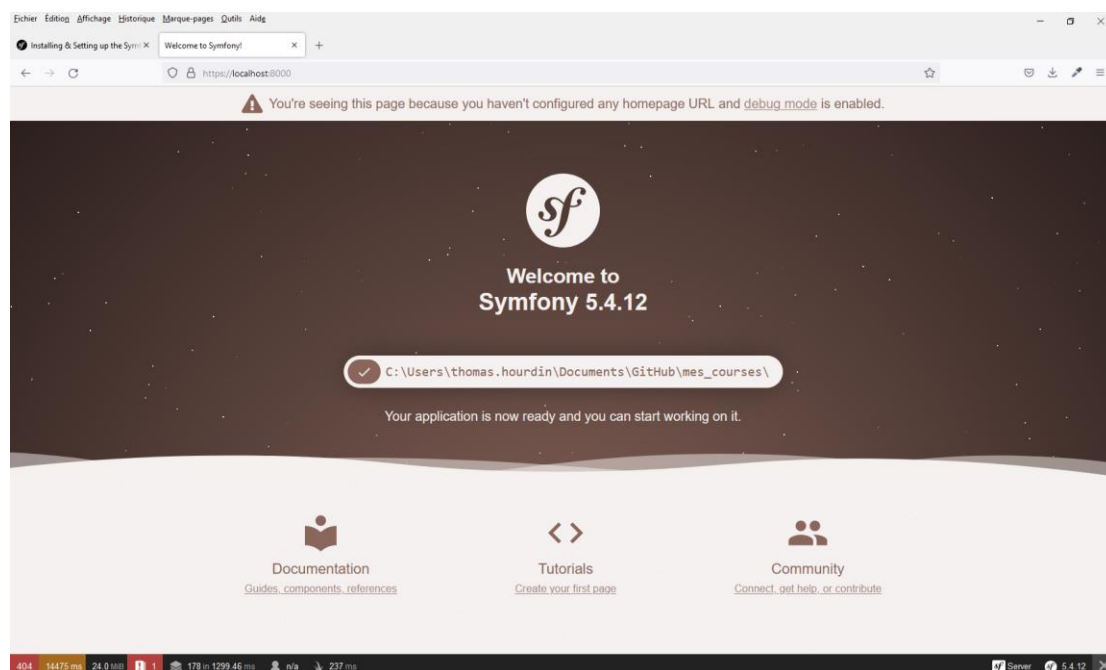
[OK] Web server listening
https://127.0.0.1:8000

Stream the logs via symfony.exe server:log
C:\Users\installateur\Documents>
```

Autorisez l'accès à `symfony.exe` dans le pare-feu :



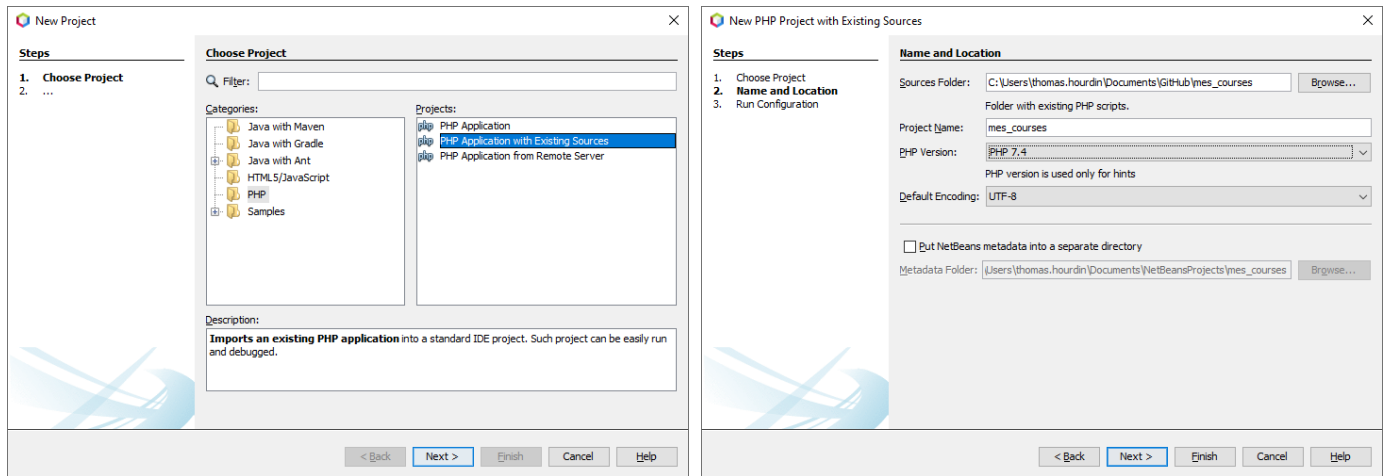
Votre serveur est démarré ! Vous pouvez maintenant accéder à votre projet dans votre navigateur Web à l'adresse indiquée : <https://127.0.0.1:8000>



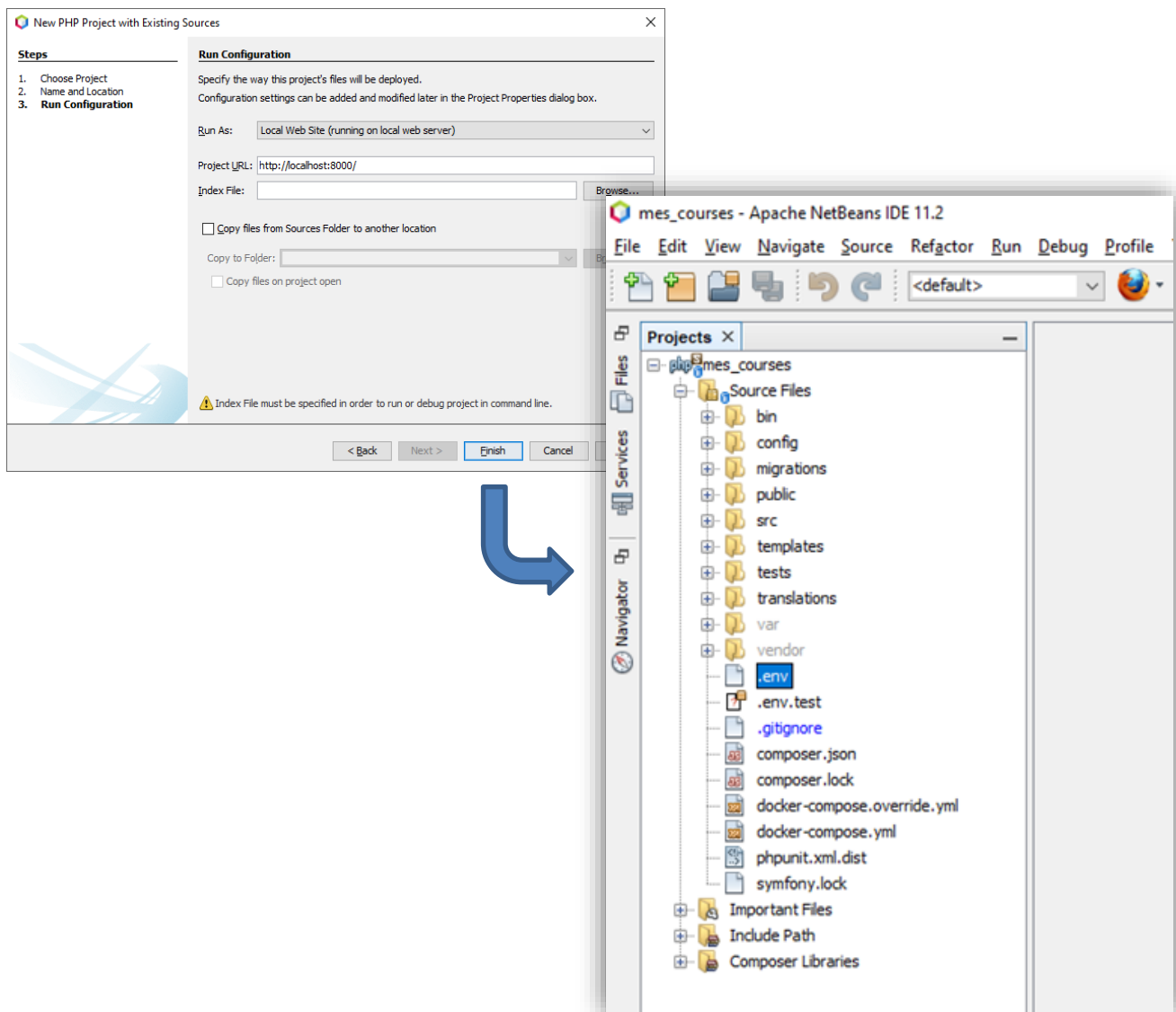
Création d'un projet *Php* sous *NetBeans*

Lancez *NetBeans*, et créez un projet PHP avec des sources existantes.

Allez chercher le répertoire contenant votre projet, et choisissez la version la plus récente de PHP.



Une fois créé, voici l'arborescence de votre projet :

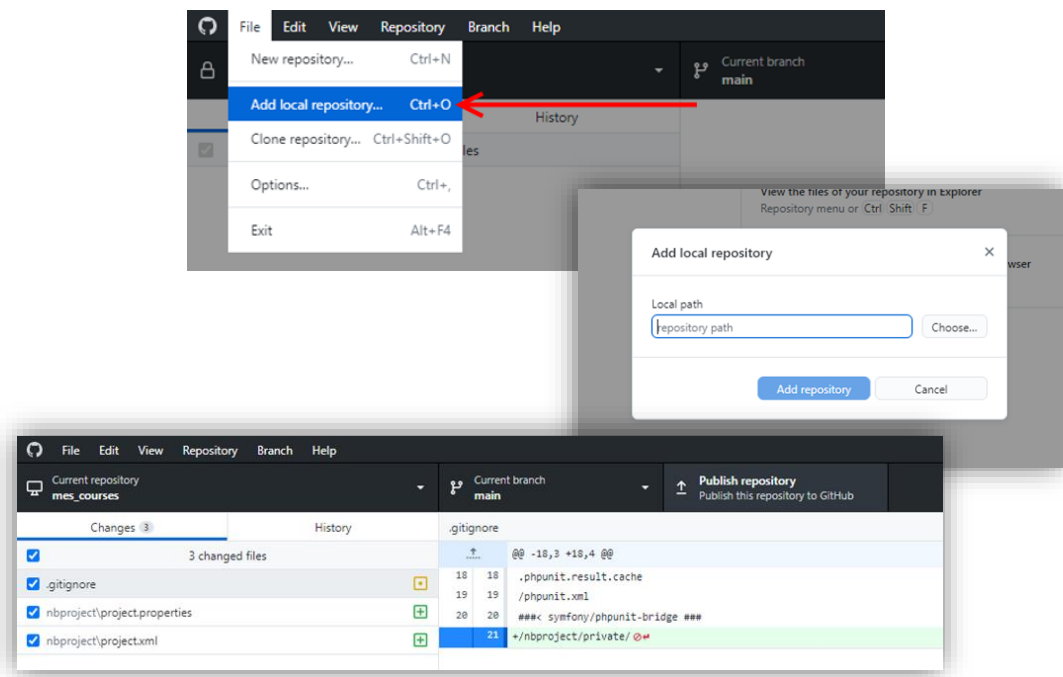


Travailler avec GitHub

GitHub est un outil incontournable lorsque vous travaillez sur un projet de développement logiciel, seul ou en équipe. Nous l'étudierons dans le cadre d'un autre cours. Pour cette étude, nous nous contenterons de créer un dépôt local avec *GitHub Desktop*.

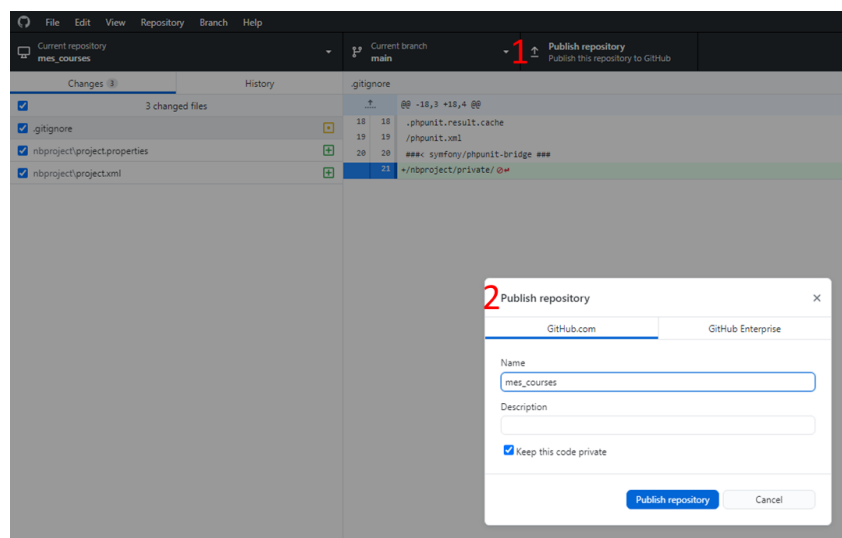
Créer un dépôt local

Lancez *GitHub Desktop* et créez un nouveau dépôt local.



Publier votre dépôt

Une fois créé, publiez votre dépôt.



Une copie de votre projet *mes_courses* est dorénavant sauvegardée sur votre espace GitHub, vous pourrez la récupérer sur un autre poste, consulter l'historique de vos modifications, revenir à des versions antérieures si vous faites des erreurs, ...

Le M.V.C. (Modèle - Vue - Contrôleur)

Introduction au MVC

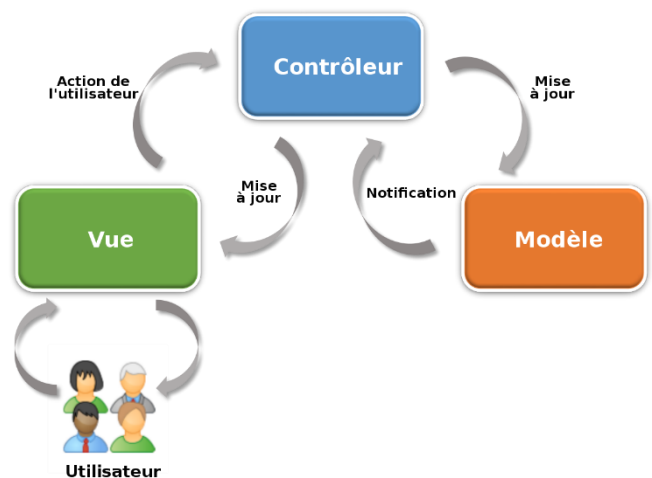
Le MVC (Modèle - Vue - Contrôleur) est un « *pattern* » qui permet une bonne organisation de votre code source.

Jusque-là, nous avons codé en PHP sans vraiment de structure. Nos pages Web mélangeaient traitement (PHP), accès aux données (SQL) et présentation (balises HTML). Tout en étant fonctionnel, cette organisation a ses limites pour des projets importants.

La structure MVC intègre les concepts suivants :

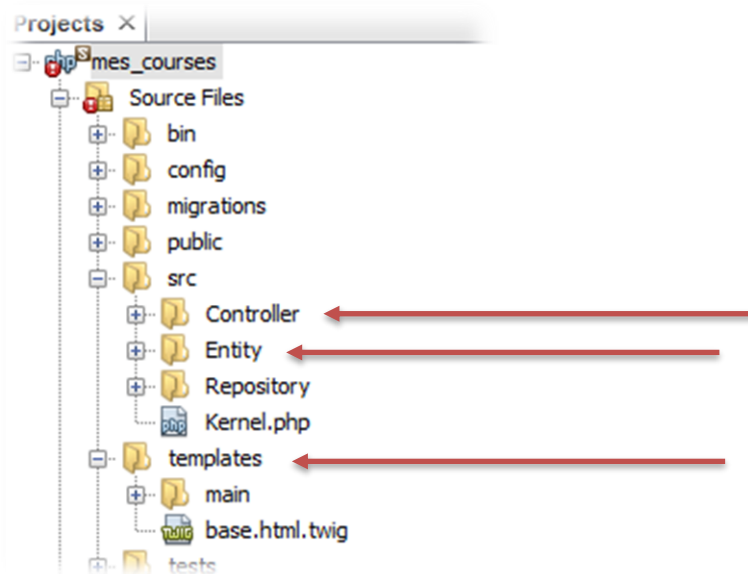
- ✓ Le **Modèle** (Accès aux données)
- ✓ La **Vue** (Interface utilisateur)
- ✓ Le **Contrôleur** (Traitement des requêtes *http*, mise en forme de la vue)

Symfony est basé sur le MVC, vous retrouverez dans l'organisation du projet les fichiers *contrôleurs*, les *templates* des pages HTML (vues) et les *entités* persistées en BDD (modèles).



Organisation des fichiers de votre projet

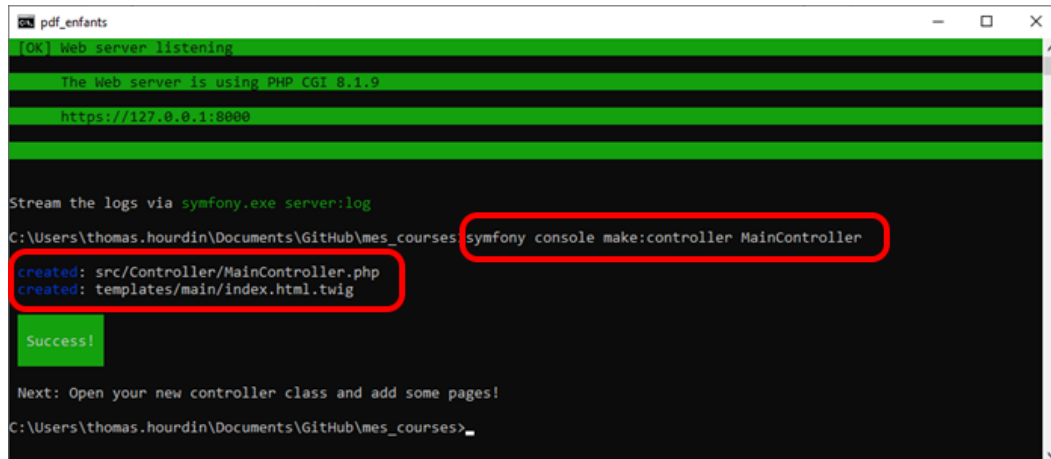
On retrouve ces 3 éléments dans l'arborescence de notre projet.



Création d'une première page

Le contrôleur « MainController »

Dans la console, créez utilisez la commande : `symfony console make :controller MainController`



```
pdf_enfants
[OK] Web server listening
The Web server is using PHP CGI 8.1.9
https://127.0.0.1:8000

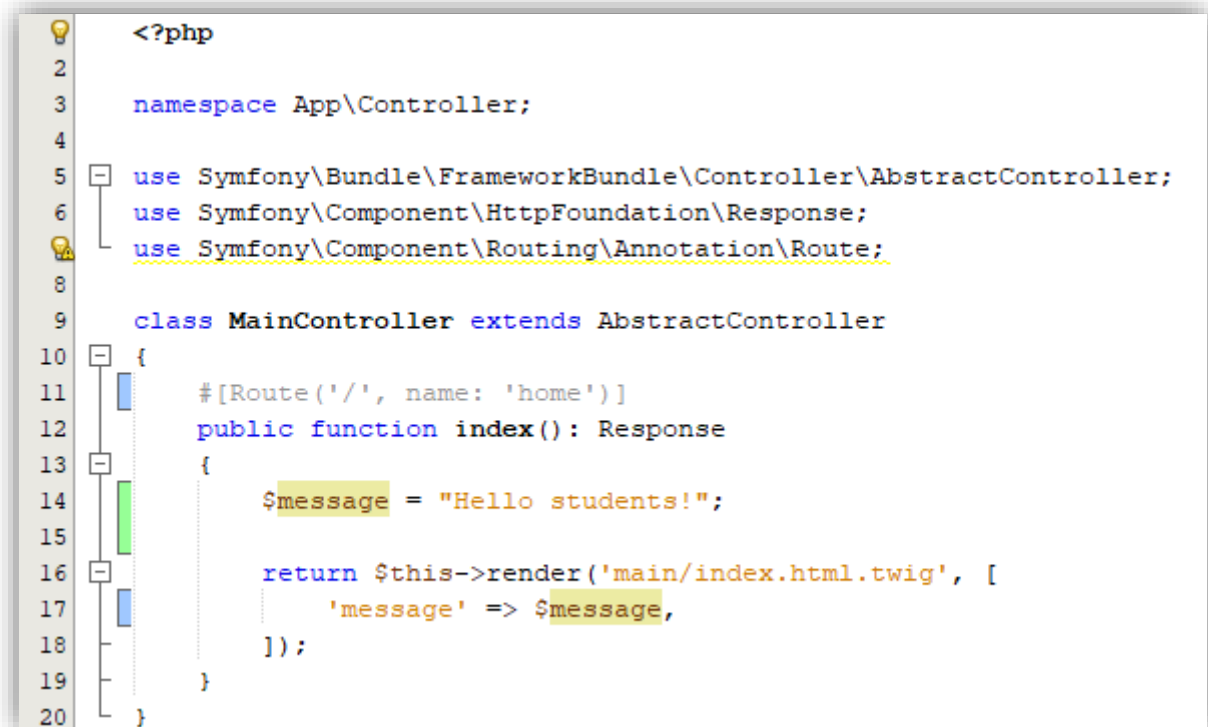
Stream the logs via symfony.exe server:log
C:\Users\thomas.hourdin\Documents\GitHub\mes_courses> symfony console make:controller MainController
created: src/Controller/MainController.php
created: templates/main/index.html.twig
Success!

Next: Open your new controller class and add some pages!
C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>
```

2 nouveaux fichiers sont créés (un contrôleur et une vue).

Création d'une action (méthode) « index »

Editez le fichier du contrôleur dans NetBeans, et ajoutez la fonction « `index()` » suivante :



```
<?php
2
3 namespace App\Controller;
4
5 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
6 use Symfony\Component\HttpFoundation\Response;
7 use Symfony\Component\Routing\Annotation\Route;
8
9 class MainController extends AbstractController
10 {
11     #[Route('/', name: 'home')]
12     public function index(): Response
13     {
14         $message = "Hello students!";
15
16         return $this->render('main/index.html.twig', [
17             'message' => $message,
18         ]);
19     }
20 }
```

La route 'home'

Au-dessus de la méthode, en commentaire, vous avez la déclaration de *la route* qui mènera à cette fonction.

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

'home' est le nom de la route, et '/' est son chemin (<https://domaine-du-projet/>). Comme cette route dirige vers la page d'accueil, son chemin est des plus basiques.

Appel du template 'index.html.twig' pour générer la page à créer

A la fin de l'action *index()*, la syntaxe suivante va appeler le template *index.html.twig* pour « rendre » la vue. La variable « message » est passée à la vue, elle contient le contenu de la variable php *\$message*.

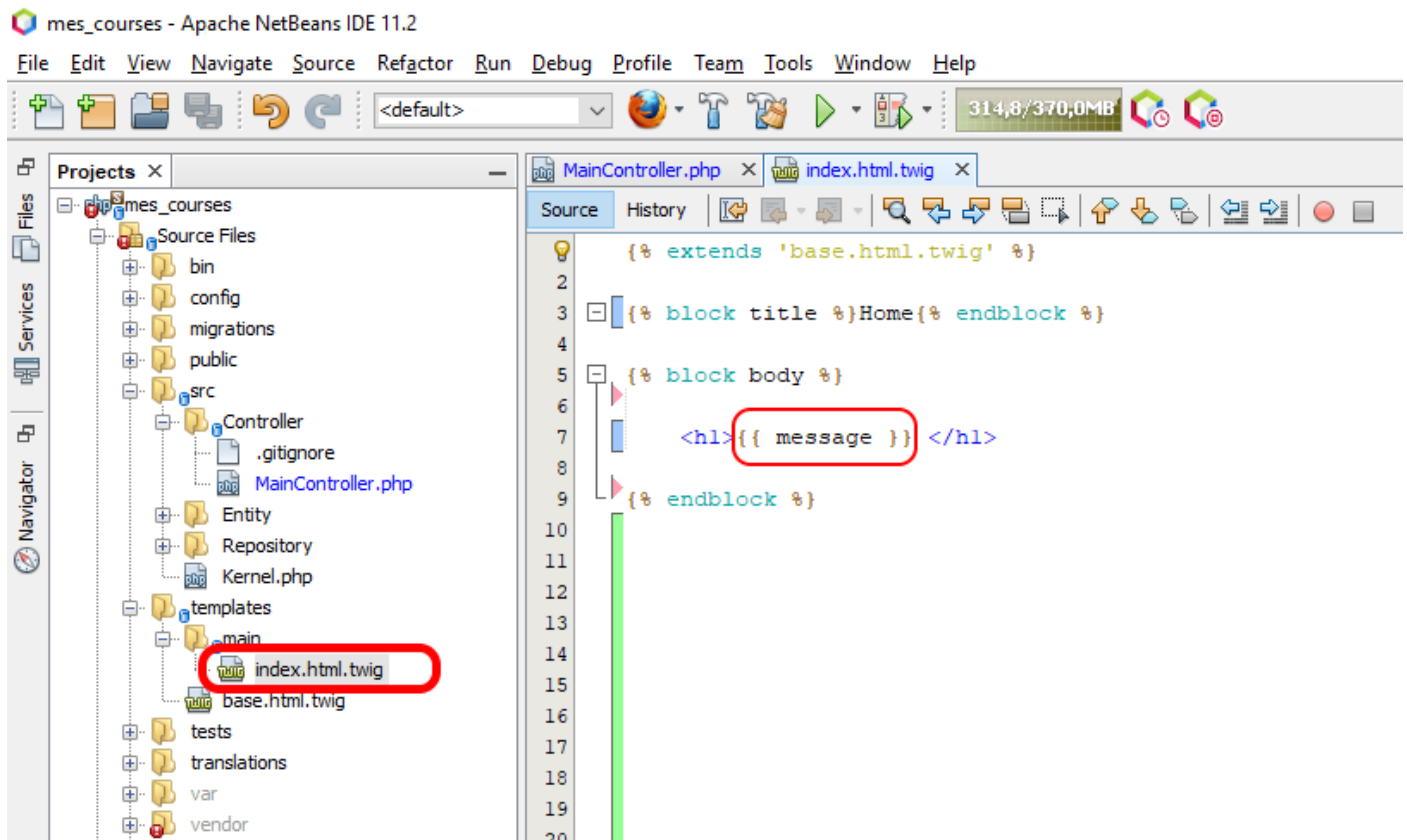
```
return $this->render('main/index.html.twig', [  
    'message' => $message,  
]);
```

Le template TWIG

Le template *index.html.twig* est une page html dans laquelle des instructions *TWIG* sont ajoutées.

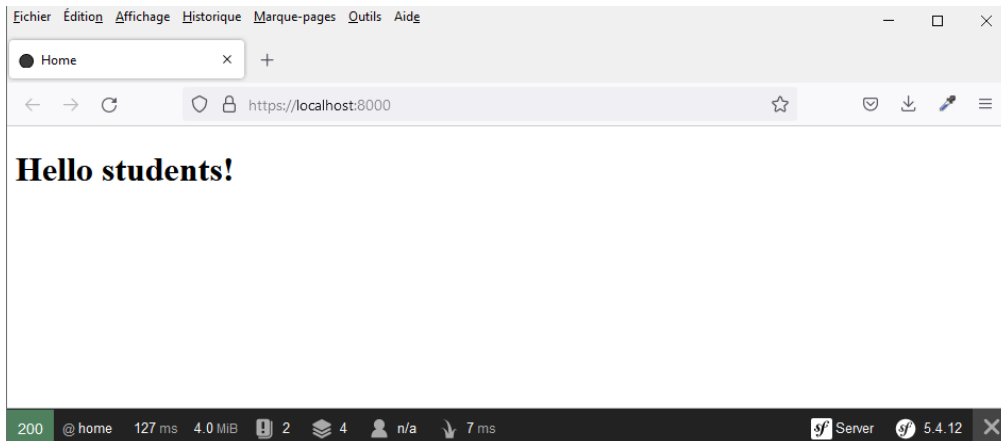
TWIG est un nouveau langage qui comme le *PHP* s'ajoute au *HTML* pour être interprété côté serveur. Sa syntaxe est assez simple et facile à distinguer.

Reprenez le code suivant pour afficher le message envoyé par le contrôleur.



Résultat de notre première page Home

A ce stade, si vous rafraîchissez la page d'accueil, vous obtiendrez ceci :



Intégration de *Bootstrap*

Il est intéressant d'intégrer bootstrap à notre projet Symfony pour tous les avantages qu'on connaît. Même si la démarche pour l'installer aujourd'hui consiste à passer par le [Webpack Encore](#), nous opterons pour une démarche plus simple et très rapide.

La version simple : Partir du template Bootstrap

Ouvrez le template de base de votre projet (fichier *base.html.twig*), et remplacez son contenu par le *template de base Bootstrap* que vous trouverez sur le site officiel :

<https://getbootstrap.com/docs/5.2/getting-started/introduction/>

Prenez soin d'y intégrer les balises TWIG qui seront redéfinies dans les pages de notre projet.

Assurez-vous d'avoir le lien vers le CSS et le lien vers le JAVASCRIPT de Bootstrap pour que tout fonctionne.

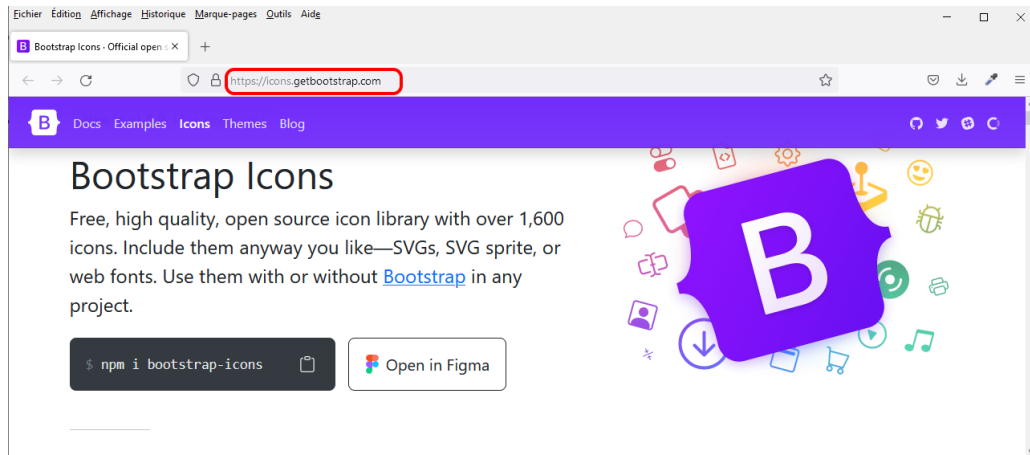
Votre template doit ressembler à ceci :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>{% block title %}Mes Courses{% endblock %}</title>
    {% block stylesheets %}{% endblock %}
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1...">
  </head>
  <body>
    {% block body %}
    {% endblock %}

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2..."></script>
    {% block javascripts %}{% endblock %}
  </body>
</html>
```

Utilisation des icones Bootstrap

Les icones *Bootstrap* sont très utiles, vous les trouverez à l'adresse <https://icons.getbootstrap.com/>.



Pour utiliser les icones de Bootstrap, intégrez le lien ci-dessous dans les CSS de votre template de base.

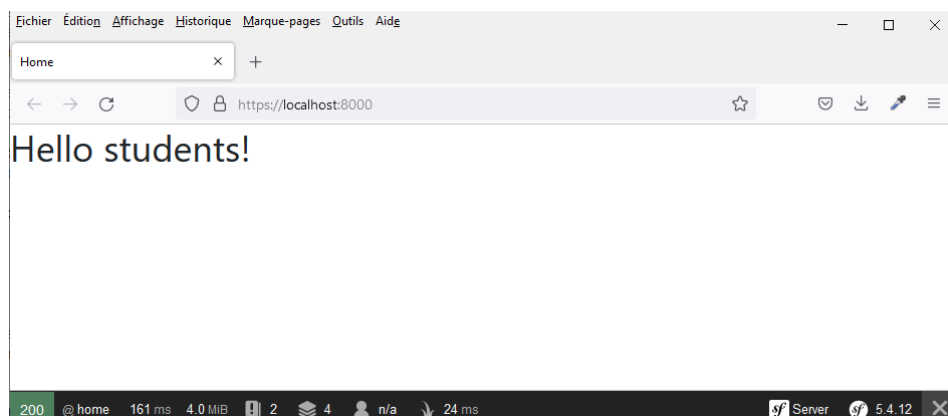
```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-icons.css">
```

Votre template ressemble maintenant à ceci :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>{% block title %}Mes Courses{% endblock %}</title>
    {% block stylesheets %}{% endblock %}
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1...>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1...>
  </head>
  <body>
    {% block body %}
    {% endblock %}

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2...></script>
    {% block javascripts %}{% endblock %}
  </body>
</html>
```

A ce stade-là, la page que nous avons créée précédemment devrait changer d'allure (la police par défaut de Bootstrap n'est plus le *Times New Roman* !).



La procédure conseillée : Utilisation du Webpack Encore

Aujourd'hui, il est conseillé d'utiliser le *Webpack Encore* pour l'intégration du CSS et JS.

La démarche consiste à installer le *Webpack Encore* via composer, puis d'installer Bootstrap avec le *gestionnaire de packages npm*. Nous ne nous attarderons pas sur cette procédure pour ce TP.

Vous trouverez toutes les informations nécessaires dans la documentation de Symfony :

<https://symfony.com/doc/5.4/frontend/encore/bootstrap.html>

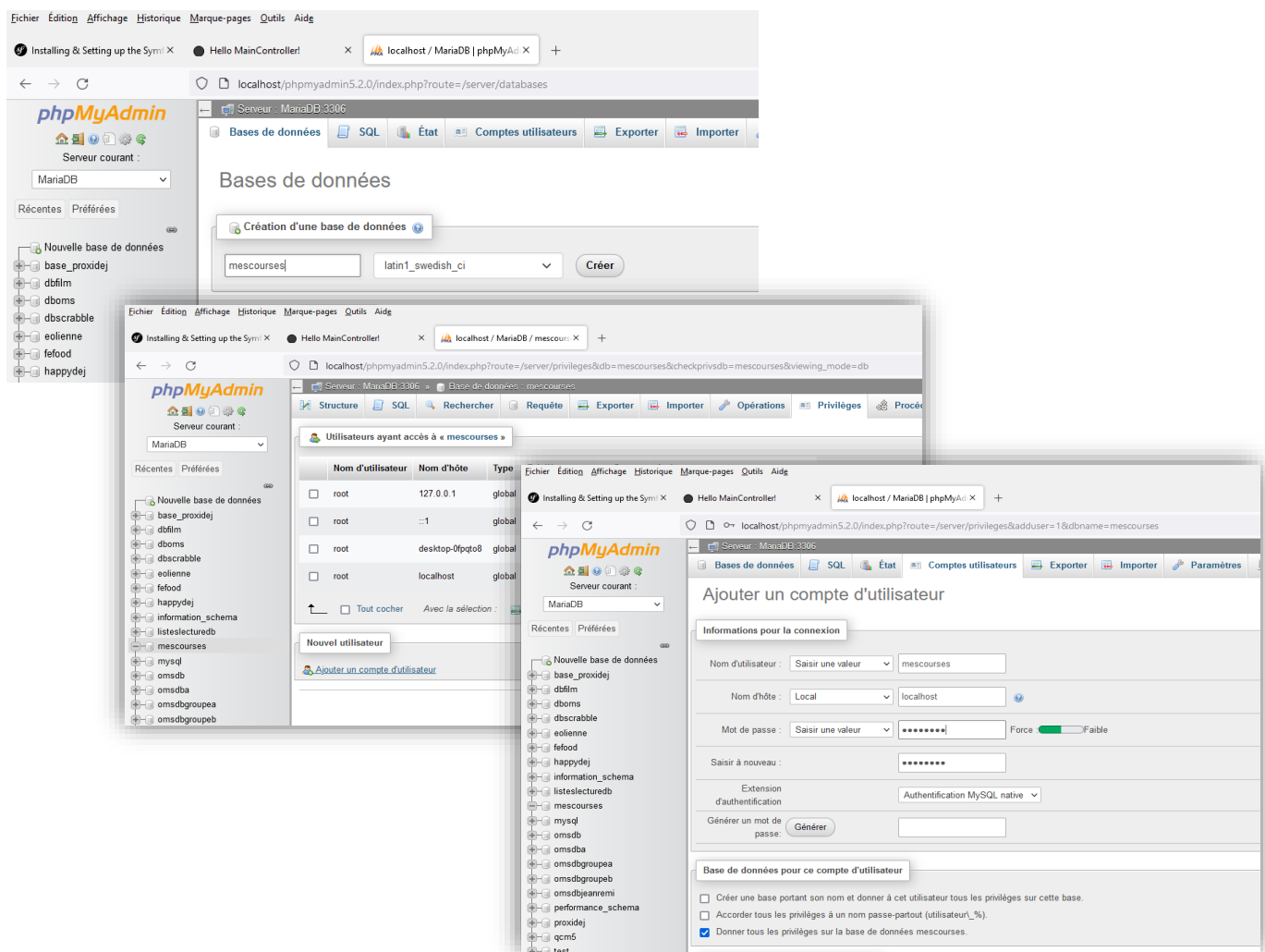
Installation de la base de données

Avant de commencer

Nous allons voir le modèle relationnel du projet que nous souhaitons développer, cependant avec Symfony, la première chose à retenir est qu'il ne faut pas réfléchir BDD mais entités.

Création de la base de données sous phpMyAdmin

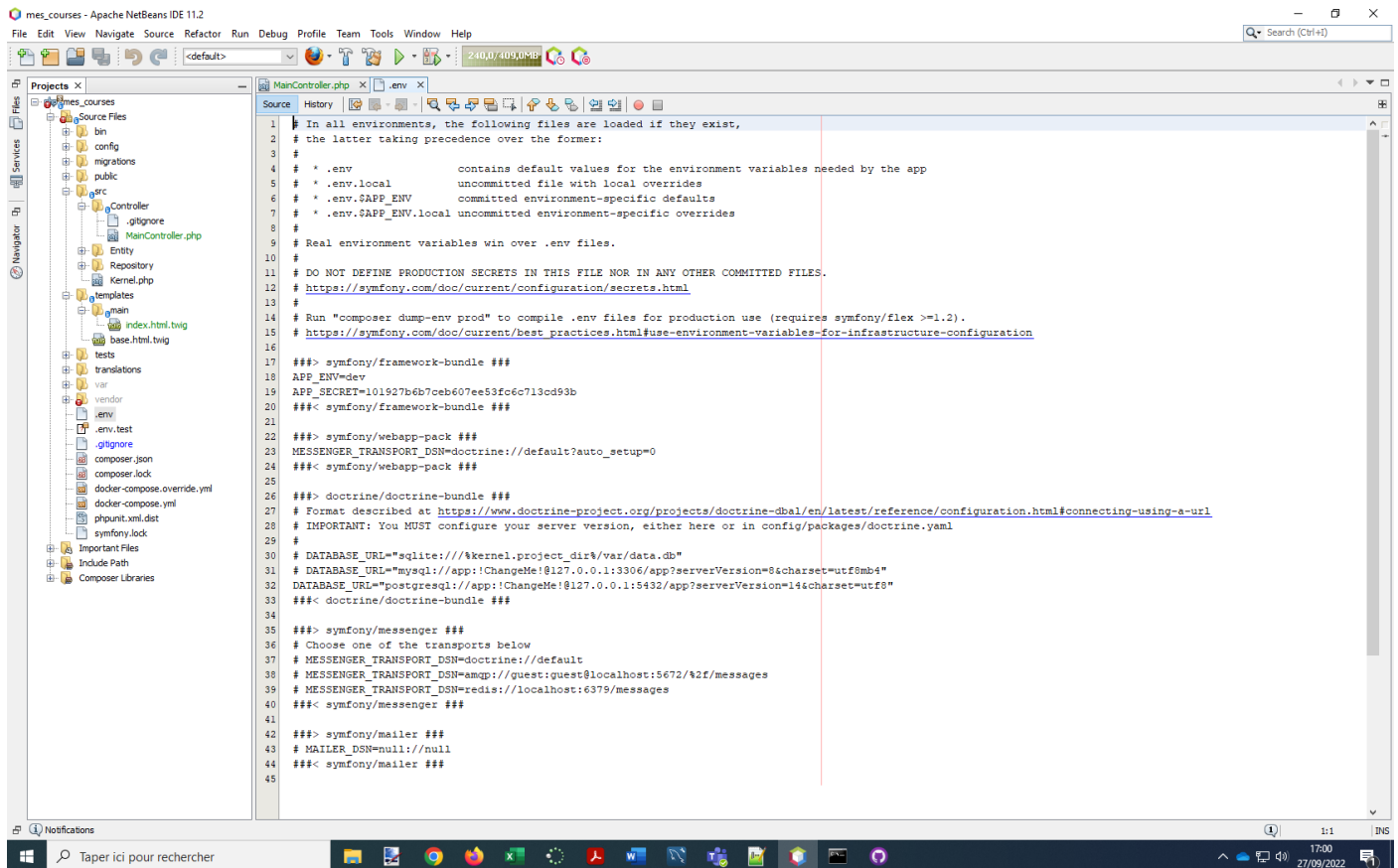
Commencez par créer une nouvelle base de données, puis créez un utilisateur spécifique pour cette base.



Configuration du fichier `.env.local`

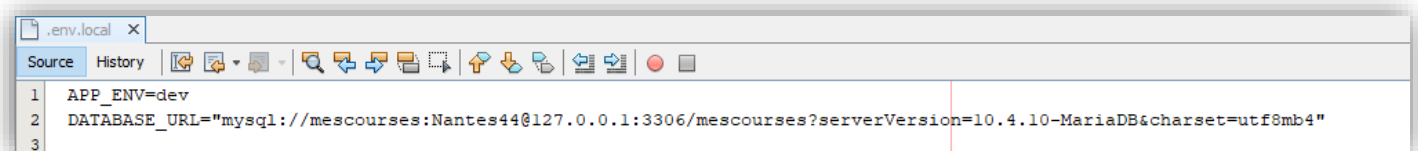
Le fichier `.env` contient les données de configuration de votre projet dans son environnement d'exploitation.

En voici un aperçu :



Pour éviter de modifier ce fichier à chaque fois que vous mettez votre projet en production, **créez un fichier `.env.local`** qui contiendra les informations de configuration de votre environnement de développement (vous ne transférez pas ce fichier en production).

Vous redéfinirez dans ce fichier uniquement les paramètres propres à votre environnement de travail local.



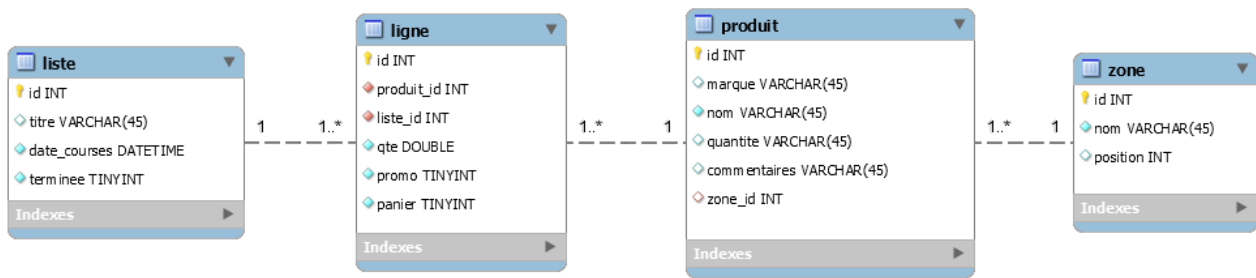
Nous nous contenterons des 2 lignes ci-dessus.

La première définit le mode développement (qui permet d'avoir la barre Symfony et le débogage), la seconde ligne définit les paramètres de connexion à la base de données.

La base de données étant correctement configurée, nous allons pouvoir passer à la création de notre première entité.

Aperçu du MCD du projet

Le modèle relationnel de notre base ressemblera à ça :



Ceci étant dit, sous Symfony vous devez faire abstraction de la base de données et raisonner « **entités** ».

Les entités

Une entité est une classe qui permettra de manipuler des objets pour les besoins de notre projet.

L'entité : Zone

Considérons une première entité **Zone** pour définir un rayon du magasin dans lequel nous avons l'habitude d'aller faire des courses. La création d'une entité **Zone** nous permettra d'organiser les items de notre liste de courses dans l'ordre de passage dans le magasin (pour éviter les aller / retours).

Qu'avons-nous besoin de renseigner ?

Nous voulons pouvoir nommer cette zone (Fruits et légumes, Pâtes/Riz, Rayon surgelés...) et lui indiquer un numéro de position sur notre itinéraire dans le magasin.

Création de l'entité

Depuis la console, créez une nouvelle entité avec l'instruction : `php bin/console make:entity Zone`

Référez-vous à l'écran ci-dessous pour compléter cette entité :

```

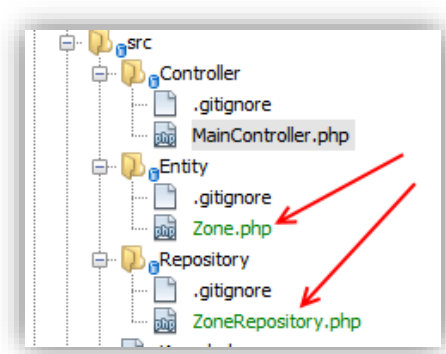
New property name (press <return> to stop adding fields):
> nom
Field type (enter ? to see all types) [string]:
>
Field length [255]:
>
Can this field be null in the database (nullable) (yes/no) [no]:
>
updated: src/Entity/Zone.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> position
Field type (enter ? to see all types) [string]:
> integer
Can this field be null in the database (nullable) (yes/no) [no]:
> yes
updated: src/Entity/Zone.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>

Success!
  
```

Une nouvelle entité et son repository sont créés dans l'arborescence de votre projet.



A ce stade l'entité existe, mais **vous devez faire une migration pour qu'elle existe dans la base de données.**

Migration vers la base de données

Exécutez la commande suivante dans la console pour **créer** la migration : `php bin/console make:migration`

```
GitHub
C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>php bin/console make:migration

Success!

Next: Review the new migration "migrations/Version20220930124810.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html

C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>
```

Puis la suivante pour **exécuter** la migration : `php bin/console doctrine:migrations:migrate`

```
GitHub
C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>php bin/console doctrine:migrations:migrate

WARNING! You are about to execute a migration in database "mescourses" that could result in schema
changes and data loss. Are you sure you wish to continue? (yes/no) [yes]:
>

[notice] Migrating up to DoctrineMigrations\Version20220930124810
[notice] finished in 268.5ms, used 22M memory, 1 migrations executed, 2 sql queries

C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>_
```

Ça y est, votre nouvelle entité existe en BDD, vous pouvez l'exploiter dans vos contrôleurs.

Vous constaterez qu'une table `doctrine_migration_versions` a été créée dans votre BDD et qu'un fichier a été créé dans le répertoire migration de votre projet. Ne cherchez pas à les supprimer, laissez *Symfony* gérer les différentes migrations.

Le fichier Zone.php

Ce fichier contient votre entité, vous pourrez rajouter des méthodes à la classe Zone si besoin.

Les fixtures

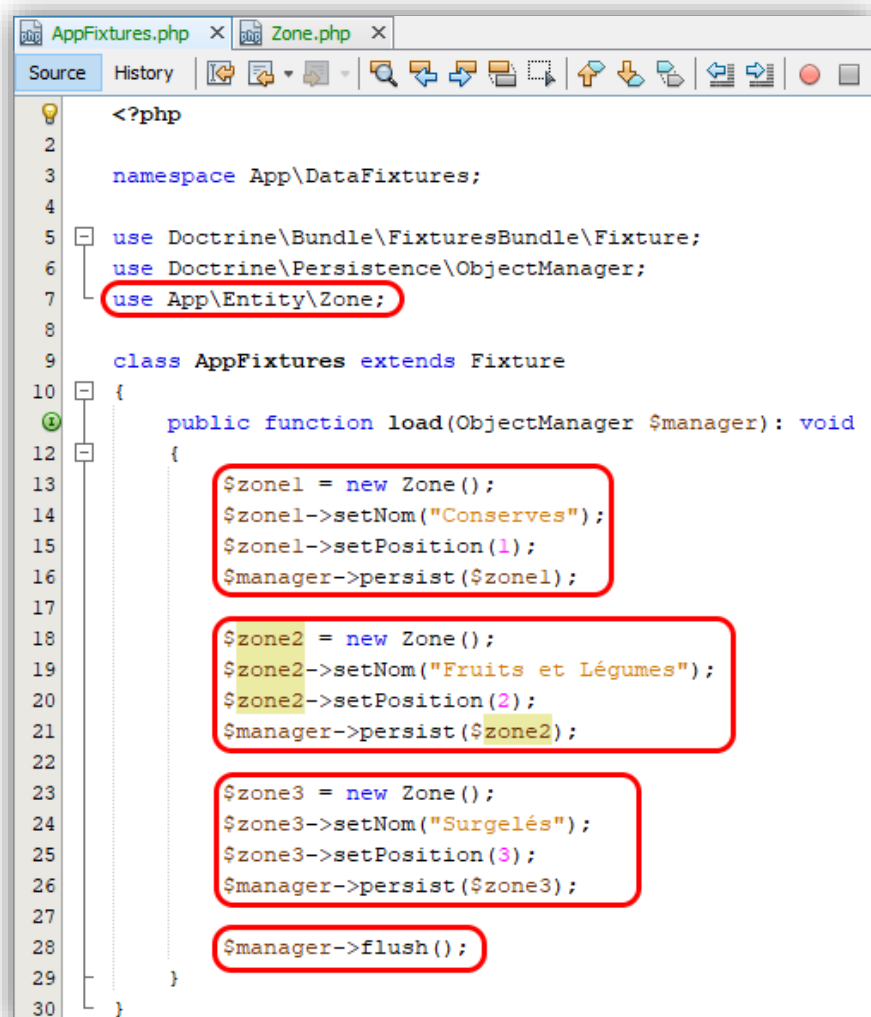
Les fixtures permettent d'insérer des enregistrements dans notre base (généralement du contenu de démonstration).

Pour cela, nous allons **installer un Bundle Fixture** dans notre projet via *Composer*.

Saisissez la commande : `composer require --dev orm-fixtures`

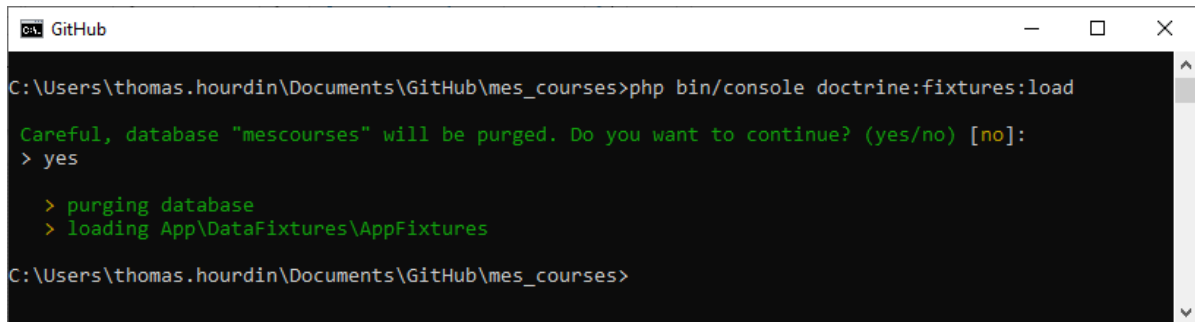
Composer va ajouter une ligne dans votre fichier *composer.json* et lancer l'installation du bundle.

Un fichier *AppFixtures* a été créé. Ouvrez-le dans *NetBeans*.



```
<?php
2
3 namespace App\DataFixtures;
4
5 use Doctrine\Bundle\FixturesBundle\Fixture;
6 use Doctrine\Persistence\ObjectManager;
7 use App\Entity\Zone;
8
9 class AppFixtures extends Fixture
10 {
11     public function load(ObjectManager $manager): void
12     {
13         $zone1 = new Zone();
14         $zone1->setNom("Conserves");
15         $zone1->setPosition(1);
16         $manager->persist($zone1);
17
18         $zone2 = new Zone();
19         $zone2->setNom("Fruits et Légumes");
20         $zone2->setPosition(2);
21         $manager->persist($zone2);
22
23         $zone3 = new Zone();
24         $zone3->setNom("Surgelés");
25         $zone3->setPosition(3);
26         $manager->persist($zone3);
27
28         $manager->flush();
29     }
30 }
```

Tapez la commande : `php bin/console doctrine:fixtures:load`



```
C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>php bin/console doctrine:fixtures:load

Careful, database "mescourses" will be purged. Do you want to continue? (yes/no) [no]:
> yes

> purging database
> loading App\DataFixtures\AppFixtures

C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>
```

Vous pouvez vérifier dans votre base de données via phpMyAdmin, des enregistrements ont été créés dans la table zone !



	id	nom	position
<input type="checkbox"/> Éditer Copier Supprimer	1	Conserves	1
<input type="checkbox"/> Éditer Copier Supprimer	2	Fruits et Légumes	2
<input type="checkbox"/> Éditer Copier Supprimer	3	Surgelés	3

Page d’affichage des Zones

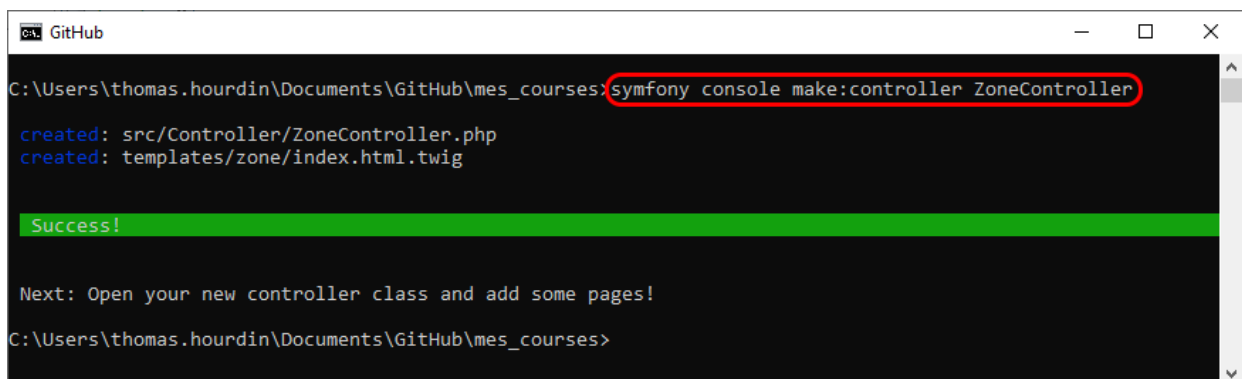
Notre objectif maintenant sera d’afficher la liste des zones sur une page Web. Nous allons procéder en 3 étapes pour y parvenir :

- ✓ Création d’un contrôleur **ZoneController**
- ✓ Création d’une action **index()** dans ce contrôleur, et définition de la route (chemin d’accès à la page)
- ✓ Création du template **index.twig.php** (la vue)

Le contrôleur **ZoneController**

Création du contrôleur

Tapez la commande : `symfony console make:controller ZoneController`



```
C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>symfony console make:controller ZoneController

created: src/Controller/ZoneController.php
created: templates/zone/index.html.twig

Success!

Next: Open your new controller class and add some pages!

C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>
```

Le contrôleur est créé, avec par défaut la création d’une action **index()**.

L'action index()

Cette action appellera la vue qui fera office de page d'accueil de la section « zones ». On souhaite que la page liste les zones existantes.

Renommez la route de l'action index() en zone_index

```
#[Route('/zone', name: 'zone_index')]
```

Nous souhaitons récupérer les zones existantes en BDD. Pour cela, nous allons utiliser le Repository **ZoneRepository**.

Un repository permet de récupérer des entités enregistrées en BDD.

Dans le contrôleur, nous allons :

- ✓ Déclarer les librairies,
- ✓ Créer un constructeur (pour avoir accès au repository dans tout le contrôleur),
- ✓ Récupérer les entités dans l'action index.
- ✓

Modifiez l'action *index()* en recopiant les lignes suivantes :

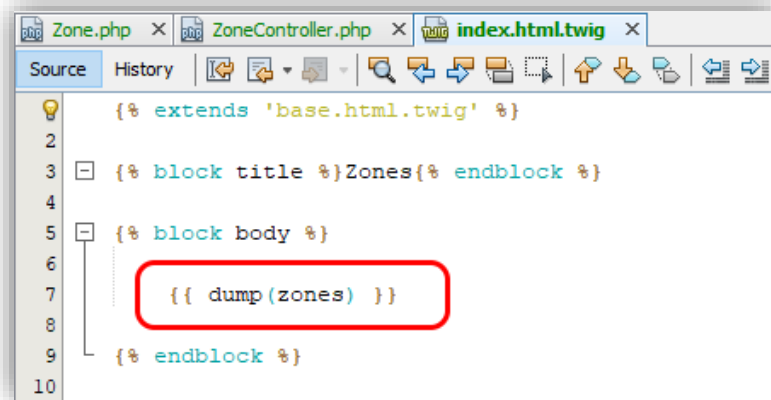
```
<?php
2
3 namespace App\Controller;
4
5 use App\Entity\Zone;
6 use App\Repository\ZoneRepository;
7
8 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
9 use Symfony\Component\HttpFoundation\Response;
10 use Symfony\Component\Routing\Annotation\Route;
11
12 #[Route('/zone')]
13 class ZoneController extends AbstractController
14 {
15     private ZoneRepository $zoneRepository;
16
17     public function __construct(ZoneRepository $zoneRepository)
18     {
19         $this->zoneRepository = $zoneRepository;
20     }
21
22     #[Route('/', name: 'zone_index')]
23     public function index(): Response
24     {
25         $zones = $this->zoneRepository->findAll();
26
27         return $this->render('zone/index.html.twig', [
28             'zones' => $zones,
29         ]);
30     }
31 }
```

Template `zone/index.twig.php`

Vous constaterez qu'à la création du contrôleur, un template a été créé pour l'action `index()`. C'est lui qui est appelé à la fin de l'action par la syntaxe :

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

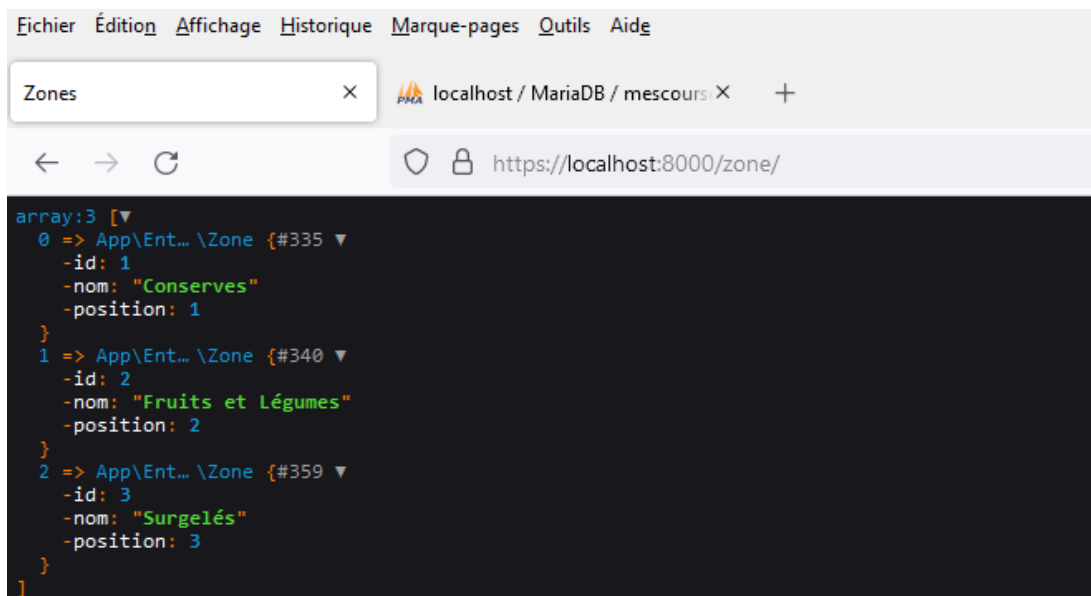
Modifier ce template pour mettre le code suivant :



Le **`dump`** permet de déboguer une variable.

Dans votre navigateur, saisissez l'URL de la route : `https://localhost:8000/zone/`

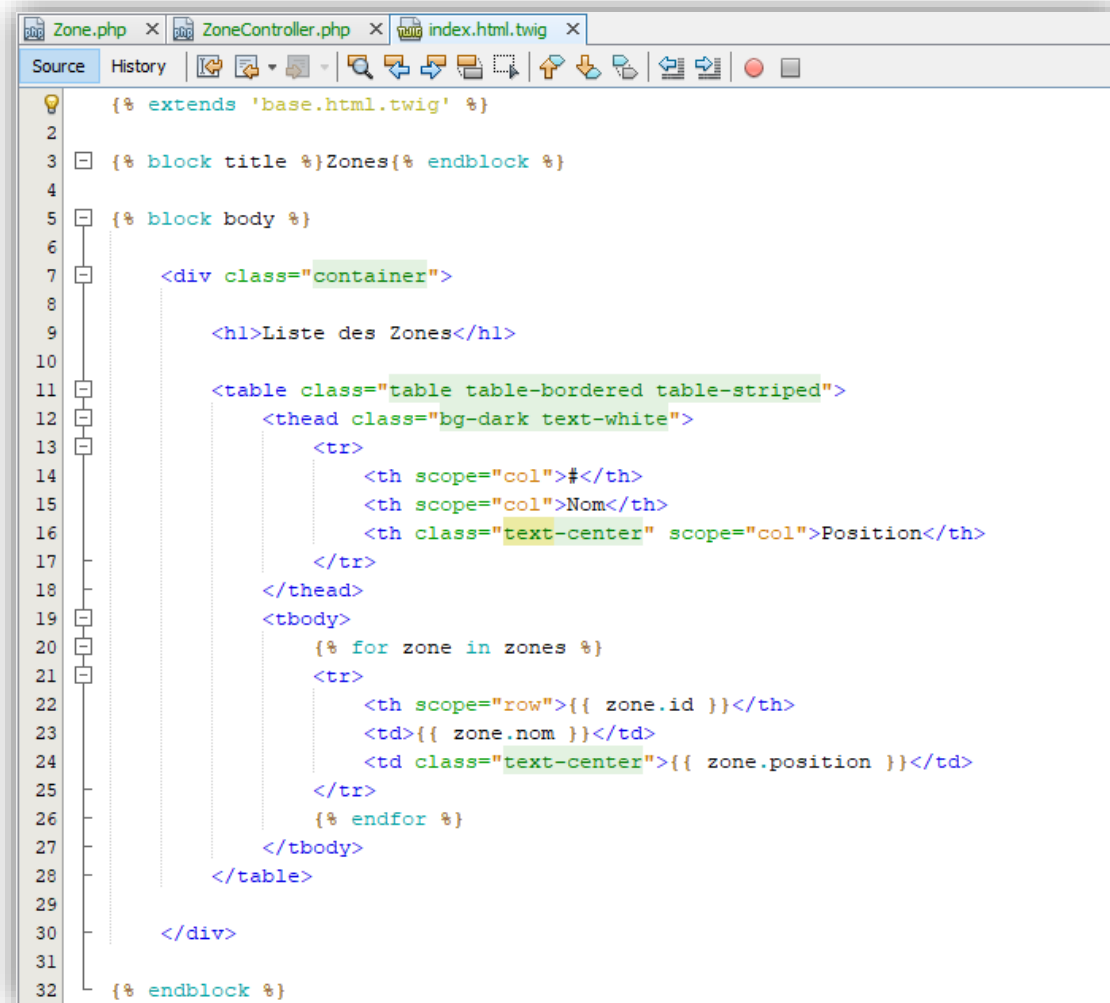
Vous devez obtenir la page suivante :



On constate bien que la variable `zones` passée à la vue est un tableau qui contient les enregistrements de la BDD.

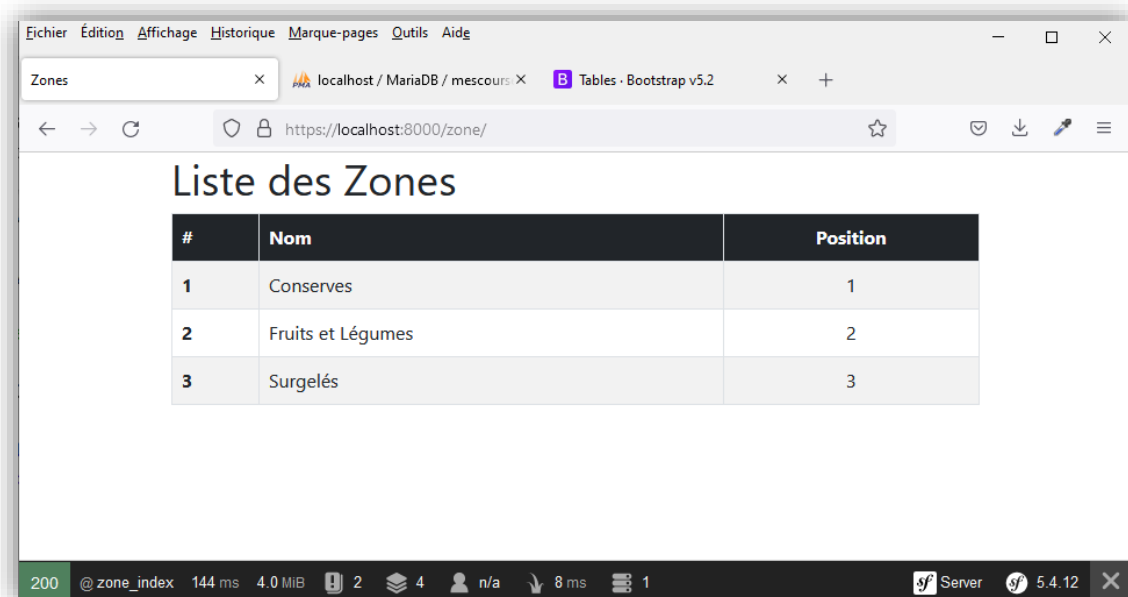
Mise en forme avec Bootstrap

Remplacez le code source du template par le suivant pour présenter les résultats dans un tableau.



```
1 {% extends 'base.html.twig' %}
2
3 {% block title %}Zones{% endblock %}
4
5 {% block body %}
6
7     <div class="container">
8
9         <h1>Liste des Zones</h1>
10
11         <table class="table table-bordered table-striped">
12             <thead class="bg-dark text-white">
13                 <tr>
14                     <th scope="col">#</th>
15                     <th scope="col">Nom</th>
16                     <th class="text-center" scope="col">Position</th>
17                 </tr>
18             </thead>
19             <tbody>
20                 {% for zone in zones %}
21                 <tr>
22                     <th scope="row">{{ zone.id }}</th>
23                     <td>{{ zone.nom }}</td>
24                     <td class="text-center">{{ zone.position }}</td>
25                 </tr>
26                 {% endfor %}
27             </tbody>
28         </table>
29
30     </div>
31
32 {% endblock %}
```

Voici le résultat :

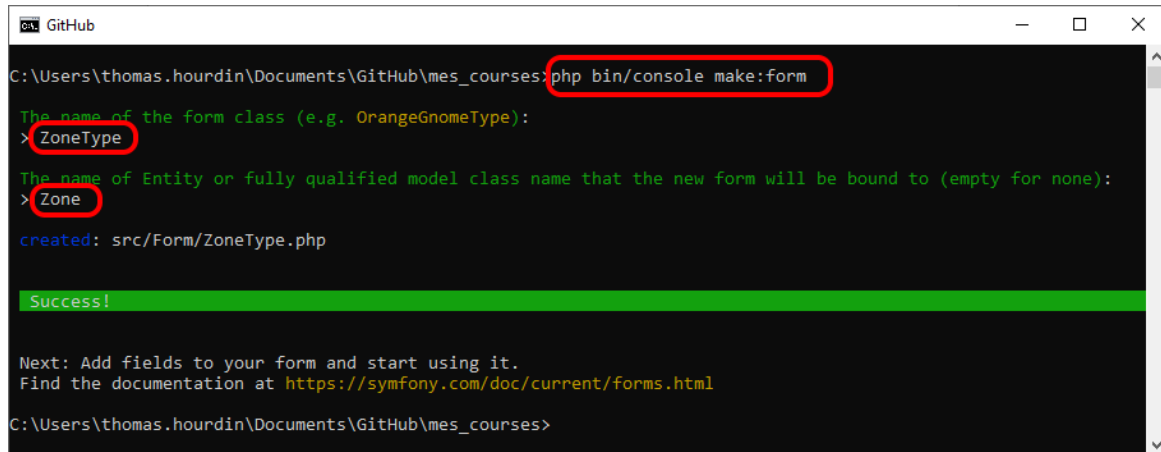


Création d'une nouvelle Zone

Génération d'un formulaire

Tapez la commande : `php bin/console make:form`

Appelez-le `ZoneType` et indiquez qu'il se base sur l'entité `Zone` (comme ci-dessous).



```
C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>php bin/console make:form

The name of the form class (e.g. OrangeGnomeType):
> ZoneType

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
> Zone

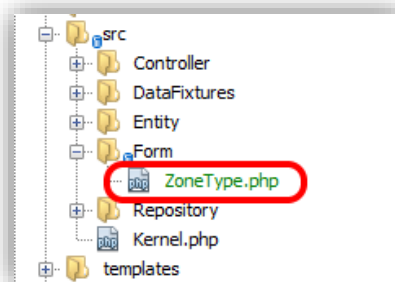
created: src/Form/ZoneType.php

Success!

Next: Add fields to your form and start using it.
Find the documentation at https://symfony.com/doc/current/forms.html

C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>
```

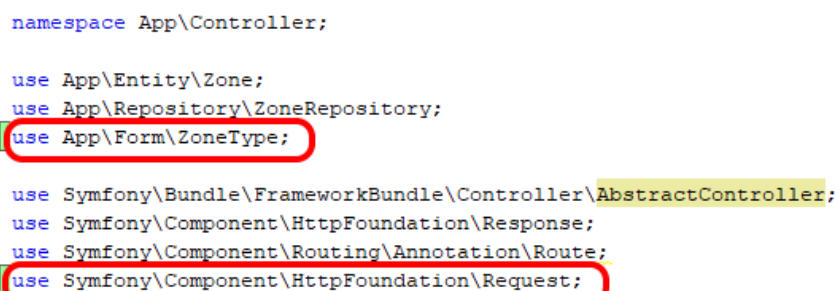
Une fois validé, le fichier `ZoneType.php` est créé dans le répertoire `Form`.



Adaptation du contrôleur

Ajout des dépendances

Avant toute chose, il faut inclure/adapter les dépendances.



```
namespace App\Controller;

use App\Entity\Zone;
use App\Repository\ZoneRepository;
use App\Form\ZoneType;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;
```

Ajout de l'action new

Nous allons maintenant créer une nouvelle action `new()` pour créer une nouvelle zone.

Recopiez le code suivant à la suite de l'action `index()`, dans la classe du contrôleur.

```
#[Route('/new', name: 'zone_new', methods: ["GET", "POST"])]
public function new(Request $request): Response {
    $zone = new Zone();

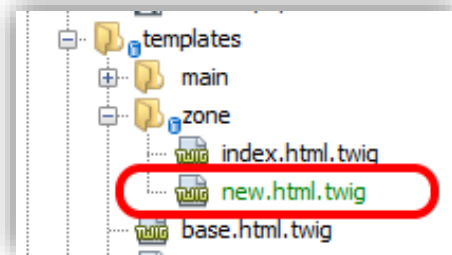
    $form = $this->createForm(ZoneType::class, $zone);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $this->zoneRepository->save($zone, true);
        return $this->redirectToRoute('zone_index', [], Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm('zone/new.html.twig', [
        'form' => $form,
    ]);
}
```

Création de la vue

Créez un fichier `new.html.twig` dans le répertoire `Template/zone` de votre projet.



Copiez/collez le code suivant :

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

{% block title %}Nouvelle Zone{% endblock %}

{% block body %}

    <div class="container">

        <h1>Nouvelle Zone</h1>

        {{ form_start(form) }}
        {{ form_widget(form) }}
        <button class="btn btn-lg btn-primary">
            {{ button_label|default('Enregistrer') }}
        </button>
        <a class="btn btn-outline-secondary" href="{{ path('zone_index') }}">Annuler</a>
        {{ form_end(form) }}

    </div>

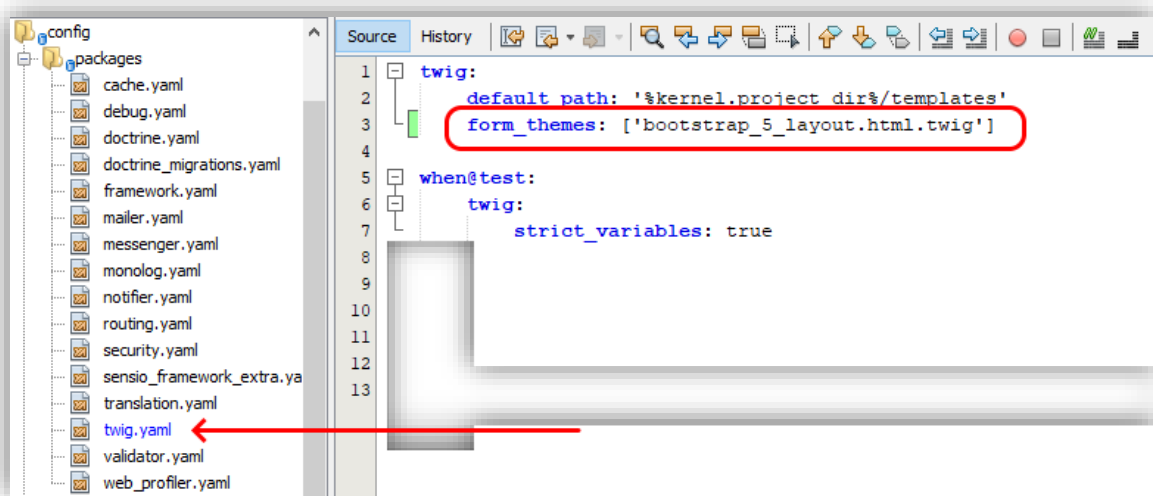
{% endblock %}
```

Vous obtenez le résultat suivant :

Et avec Bootstrap ?

Par soucis d'esthétique, nous souhaitons que les champs de formulaires soient présentés avec *Bootstrap*.

Rajoutez la ligne suivante dans le fichier `packages/twig.yaml` :



Votre formulaire sera alors un peu plus présentable, sans changer une ligne de code dans le template :

Nouvelle Zone

Nom

Position

Vous pourrez ensuite personnaliser l'affichage de cette page avec du *html* et les class *css* de *Bootstrap*.

Testez votre formulaire pour ajouter des zones, il doit fonctionner :

Liste des Zones

#	Nom	Position
1	Conserves	1
2	Fruits et Légumes	2
3	Surgelés	3
4	Entrée du magasin	0

« SI JE VEUX REORGANISER MES ZONES PAR POSITION DANS LE MAGASIN ? »

Dans le contrôleur, modifiez la requête sur le repository comme indiqué ci-dessous :

```
#[Route('/', name: 'zone_index')]
public function index(): Response
{
    $zones = $this->zoneRepository->findBy([], ['position' => 'ASC']);

    return $this->render('zone/index.html.twig', [
        'zones' => $zones,
    ]);
}
```

En rafraîchissant la page, vous obtiendrez la liste des zones triées par position croissantes.

Liste des Zones

#	Nom	Position
4	Entrée du magasin	0
1	Conserves	1
2	Fruits et Légumes	2
3	Surgelés	3

Conclusion de cette première partie

Nous allons reprendre nos templates pour :

- ✓ Ajouter une **barre de navigation** (navbar)
- ✓ Ajouter un **lien de création d'une zone**
- ✓ **Redéfinir les blocks** de notre template

Template de base : Ajout d'une Navbar et redéfinition des blocks

Nous allons modifier le template de base pour avoir une structure commune à tout l'ensemble du site. Nous définirons un bloc `main_content` que nous pourrions redéfinir pour chacune des pages.

Modifiez le template de base (fichier *base.html.twig*) comme ceci :

```
<body>
    {% block body %}
        <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
            <div class="container-fluid">
                <a class="navbar-brand" href="#">Mes Courses</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
                    data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
                    aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="collapse navbar-collapse" id="navbarNav">
                    <ul class="navbar-nav">
                        <li class="nav-item">
                            <a class="nav-link" href="{{ path('zone_index') }}">Zones</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
        <div class="container">
            {% block main_content %}
            {% endblock %}
        </div>
    {% endblock %}

    <script src="https://cdn.jsdelivr.net/..."></script>
    {% block javascripts %}{% endblock %}
</body>
```

Modifiez les 2 pages *index.html.twig* et *zone/new.html.twig* en conséquence.

Lien vers la page « Nouvelle Zone »

Pour accéder plus facilement à la page d'ajout d'une zone, on peut rajouter un bouton « Nouvelle Zone »

Voici le template final de la page *index.html.twig* :

```
{% extends 'base.html.twig' %}
{% block title %}Zones{% endblock %}
{% block main_content %}

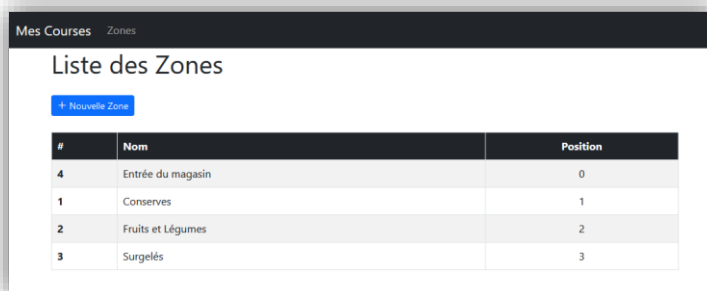
    <h1>Liste des Zones</h1>

    <p class="my-4">
        <a class="btn btn-sm btn-primary" href="{{ path('zone_new') }}"
            title="Nouvelle Zone"><i class="bi bi-plus-lg me-1"></i>Nouvelle Zone</a>
    </p>

    <table class="table table-bordered table-striped">
        <thead class="bg-dark text-white">
            <tr>
                <th scope="col">#</th>
                <th scope="col">Nom</th>
                <th class="text-center" scope="col">Position</th>
            </tr>
        </thead>
        <tbody>
            {% for zone in zones %}
            <tr>
                <th scope="row">{{ zone.id }}</th>
                <td>{{ zone.nom }}</td>
                <td class="text-center">{{ zone.position }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>

{% endblock %}
```


Résultat de l’affichage de cette page :

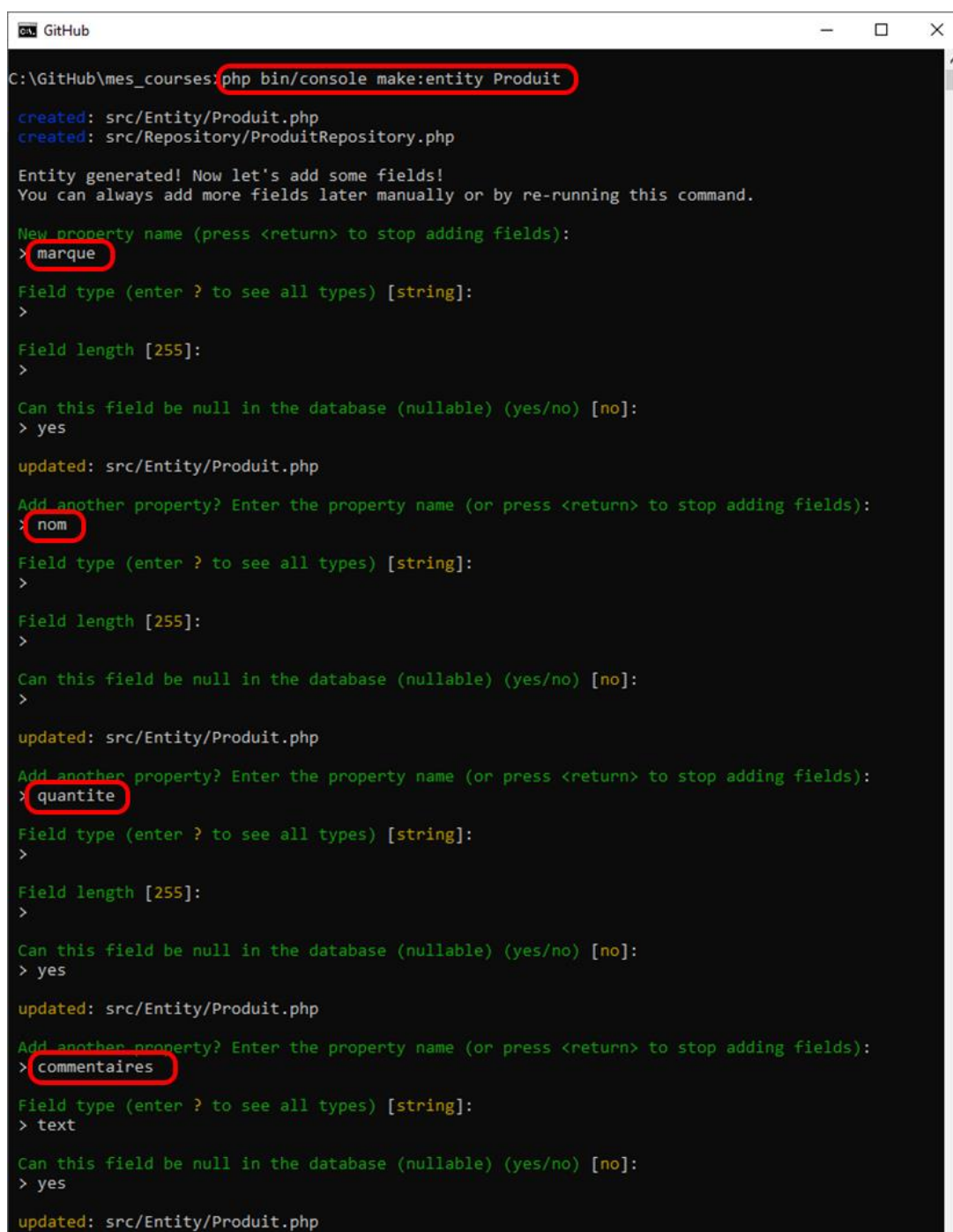


#	Nom	Position
4	Entrée du magasin	0
1	Conserves	1
2	Fruits et Légumes	2
3	Surgelés	3

L’entité *Produit*

Création de l’entité

Suivez les étapes ci-dessous pour créer l’entité *Produit*.



```
C:\GitHub\mes_courses: php bin/console make:entity Produit

created: src/Entity/Produit.php
created: src/Repository/ProduitRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> marque
Field type (enter ? to see all types) [string]:
>
Field length [255]:
>
Can this field be null in the database (nullable) (yes/no) [no]:
> yes
updated: src/Entity/Produit.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> nom
Field type (enter ? to see all types) [string]:
>
Field length [255]:
>
Can this field be null in the database (nullable) (yes/no) [no]:
>
updated: src/Entity/Produit.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> quantite
Field type (enter ? to see all types) [string]:
>
Field length [255]:
>
Can this field be null in the database (nullable) (yes/no) [no]:
> yes
updated: src/Entity/Produit.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> commentaires
Field type (enter ? to see all types) [string]:
> text
Can this field be null in the database (nullable) (yes/no) [no]:
> yes
updated: src/Entity/Produit.php
```

... la suite...

```

Add another property? Enter the property name (or press <return> to stop adding fields):
> zone

Field type (enter ? to see all types) [string]:
> relation

What class should this entity be related to?:
> Zone

What type of relationship is this?
-----
Type      Description
-----
ManyToOne Each Produit relates to (has) one Zone.
          Each Zone can relate to (can have) many Produit objects
OneToMany Each Produit can relate to (can have) many Zone objects.
          Each Zone relates to (has) one Produit
ManyToMany Each Produit can relate to (can have) many Zone objects.
          Each Zone can also relate to (can also have) many Produit objects
OneToOne  Each Produit relates to (has) exactly one Zone.
          Each Zone also relates to (has) exactly one Produit.
-----

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToOne

Is the Produit.zone property allowed to be null (nullable)? (yes/no) [yes]:
>

Do you want to add a new property to Zone so that you can access/update Produit objects from it
- e.g. $zone->getProduits()? (yes/no) [yes]:
>

A new property will also be added to the Zone class so that you can access the related Produit
objects from it.

New field name inside Zone [produits]:
>

updated: src/Entity/Produit.php
updated: src/Entity/Zone.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>

Success!

Next: When you're ready, create a migration with php bin/console make:migration

C:\GitHub\mes_courses>

```

Migration de la BDD

Maintenant que cette nouvelle entité est créée, il faut la persister en base de données.

Créez une migration pour prendre en compte la nouvelle entité :

```
php bin/console make:migration
```

Exécutez la migration :

```
php bin/console doctrine:migrations:migrate
```

La base de données a maintenant une nouvelle table « produit » !

```

C:\GitHub\mes_courses> php bin/console make:migration

Success!

Next: Review the new migration "migrations/Version20221006150100.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html

C:\GitHub\mes_courses> php bin/console doctrine:migrations:migrate

WARNING! You are about to execute a migration in database "mescourses" that could result
in schema changes and data loss. Are you sure you wish to continue? (yes/no) [yes]:
>

[notice] Migrating up to DoctrineMigrations\Version20221006150100
[notice] finished in 142.7ms, used 22M memory, 1 migrations executed, 2 sql queries

C:\GitHub\mes_courses>

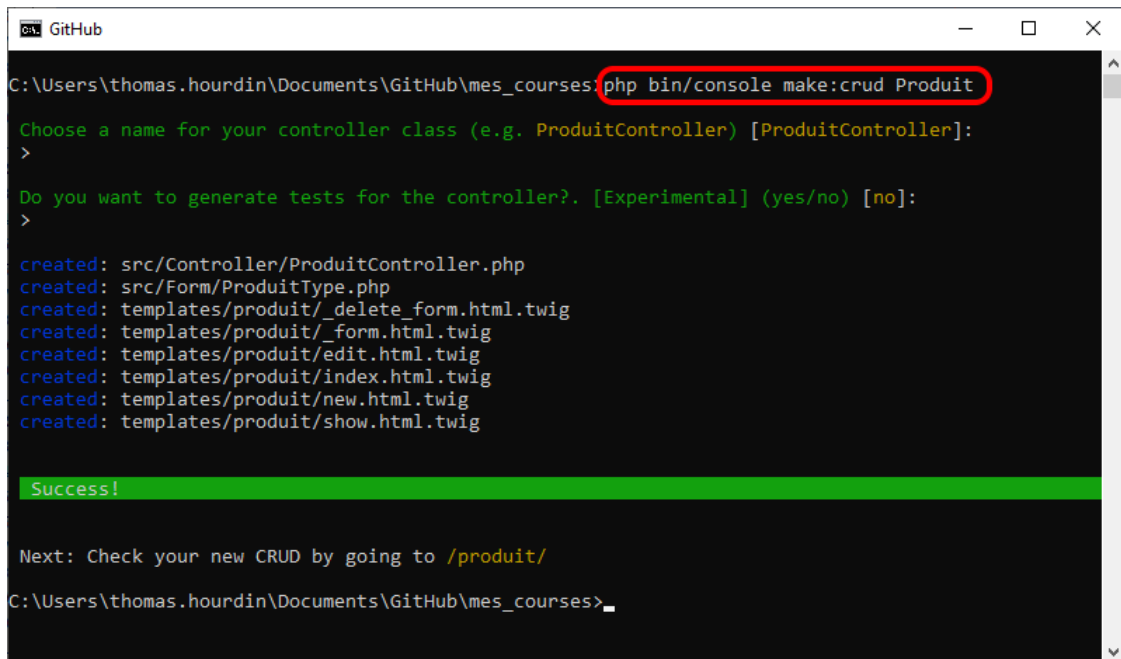
```

Gagner du temps avec le CRUD !

CRUD est l'acronyme pour *Create, Read, Update, Delete*. Ce sont les fonctions de base que l'on souhaite généralement effectuer sur les entités.

Une commande *Symfony* permet de nous faire gagner beaucoup de temps : `php bin/console make:crud`

Utilisons la pour l'entité *Produit* que nous venons de créer :



```
C:\Users\thomas.hourdin\Documents\GitHub\mes_courses> php bin/console make:crud Produit

Choose a name for your controller class (e.g. ProduitController) [ProduitController]:
>

Do you want to generate tests for the controller?. [Experimental] (yes/no) [no]:
>

created: src/Controller/ProduitController.php
created: src/Form/ProduitType.php
created: templates/produit/_delete_form.html.twig
created: templates/produit/_form.html.twig
created: templates/produit/edit.html.twig
created: templates/produit/index.html.twig
created: templates/produit/new.html.twig
created: templates/produit/show.html.twig

Success!

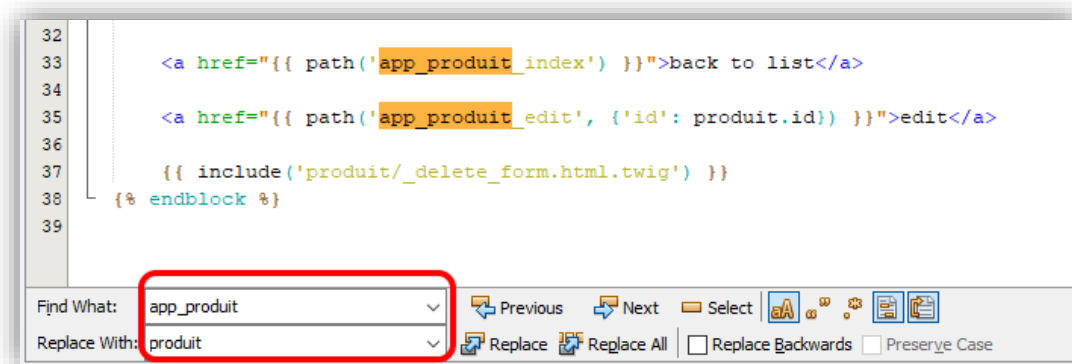
Next: Check your new CRUD by going to /produit/

C:\Users\thomas.hourdin\Documents\GitHub\mes_courses>
```

Cette commande prépare pour nous : Le *contrôleur*, le *formulaire* et les *vues* associées.

Pour être cohérent avec les routes initialement créées, nous allons supprimer le `app_` devant les noms de route.

Ouvrez tous les fichiers nouvellement créés, et faites un Ctr+H



```
32
33
34 <a href="{{ path('app_produit_index') }}">back to list</a>
35
36 <a href="{{ path('app_produit_edit', {'id': produit.id}) }}">edit</a>
37
38 {{ include('produit/_delete_form.html.twig') }}
39 {% endblock %}
```

Find What: app_produit
Replace With: produit

Validez cette modification pour chacune des pages ouvertes.

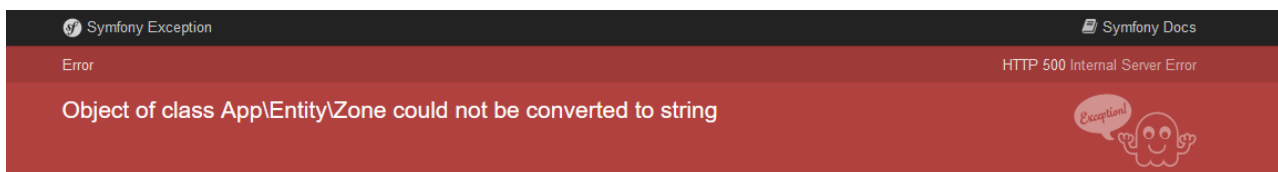
Attention ! La commande précédente ne fait pas non plus tout le travail.

Par exemple : La page index des produits donne le rendu suivant.

Produit index					
Id	Marque	Nom	Quantité	Commentaires	actions
no records found					
Create new					

⇒ Les blocs de notre template ne sont pas pris en compte.

La page *new* donne l'erreur suivante :



⇒ Le formulaire ne gère pas correctement la zone d'un produit.

Adaptation du CRUD

La page `index.html.twig`

Remplacez le block `body` par le block `main_content`, et traduisez la syntaxe anglaise en français.

Vous devriez obtenir le rendu suivant :

Mes Courses Zones Produits					
Liste des Produits					
Id	Marque	Nom	Quantité	Commentaires	actions
Aucun produit enregistré					
Nouveau Produit					

Le formulaire ProduitType

Le formulaire proposé ressemble à ceci :

```
namespace App\Form;

use App\Entity\Produit;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ProduitType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('marque')
            ->add('nom')
            ->add('quantite')
            ->add('commentaires')
            ->add('zone')
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Produit::class,
        ]);
    }
}
```

Pour améliorer cette version, remplacez le contenu de ce fichier par le code suivant :

```
namespace App\Form;

use App\Entity\Produit;
use App\Entity\Zone;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;

class ProduitType extends AbstractType {

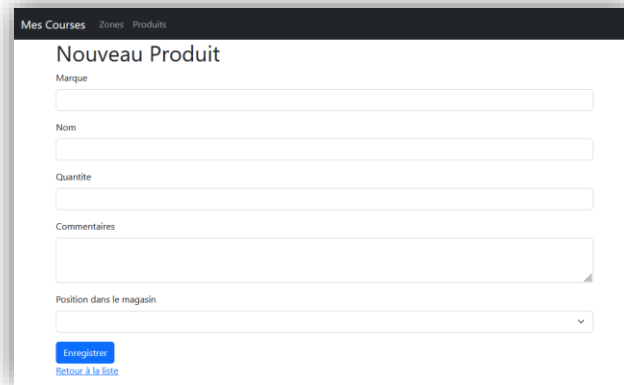
    public function buildForm(FormBuilderInterface $builder, array $options): void {
        $builder
            ->add('marque', TextType::class, [
                'required' => false,
            ])
            ->add('nom', TextType::class, [
            ])
            ->add('quantite', TextType::class, [
                'required' => false,
            ])
            ->add('commentaires', TextareaType::class, [
                'required' => false,
            ])
            ->add('zone', EntityType::class, [
                'label' => "Position dans le magasin",
                'class' => Zone::class,
                'choice_label' => 'nom',
                'required' => false,
            ])
        ];
    }

    public function configureOptions(OptionsResolver $resolver): void {
        $resolver->setDefaults([
            'data_class' => Produit::class,
        ]);
    }
}
```

Les pages `new.html.twig` et `edit.html.twig`

Ces 2 pages utilisent le même formulaire. *Symfony* crée un fichier avec le formulaire et intègre ce fichier dans les template `new.html.twig` et `edit.html.twig`.

Adaptez ces 2 derniers fichiers pour adapter la langue et les blocks (comme pour la vue index).



Mes Courses Zones Produits

Nouveau Produit

Marque

Nom

Quantité

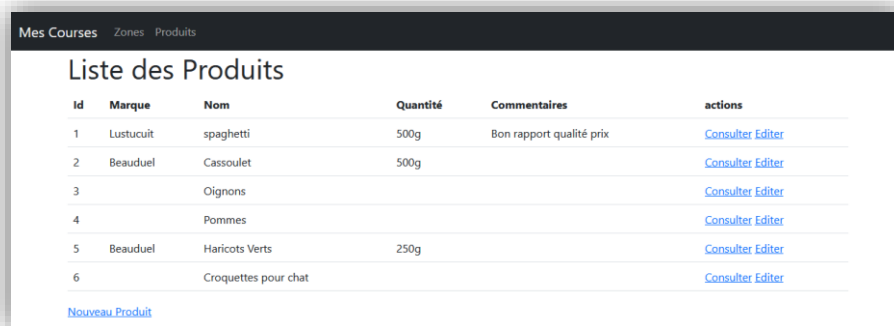
Commentaires

Position dans le magasin

[Enregistrer](#)

[Retour à la liste](#)

Vous pouvez maintenant créer quelques produits.



Mes Courses Zones Produits

Liste des Produits

Id	Marque	Nom	Quantité	Commentaires	actions
1	Lustucuit	spaghetti	500g	Bon rapport qualité prix	Consulter Editer
2	Beauduel	Cassoulet	500g		Consulter Editer
3		Oignons			Consulter Editer
4		Pommes			Consulter Editer
5	Beauduel	Haricots Verts	250g		Consulter Editer
6		Croquettes pour chat			Consulter Editer

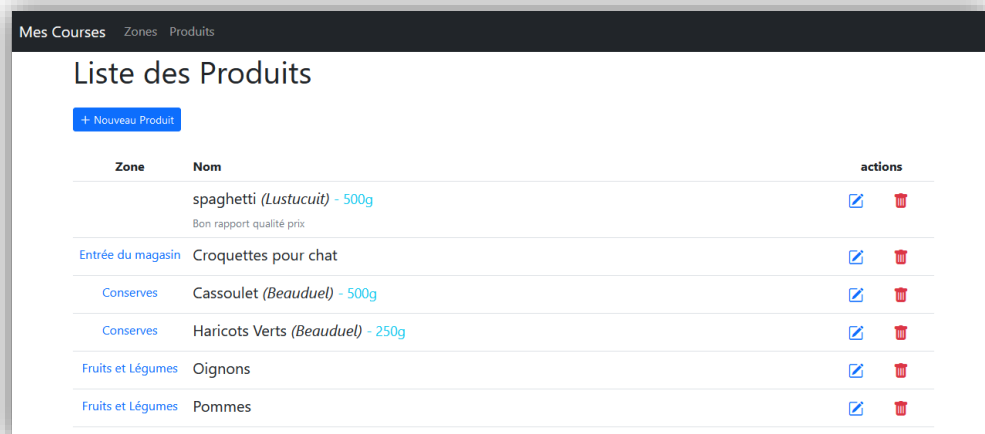
[Nouveau Produit](#)

Affichage de la liste des produits

Adaptez l’affichage de la liste des produits pour une rendu qui correspondra à vos attentes.

Pour finir, même si elle ne nous intéresse pas spécialement, ajustez la page show (vous pouvez également déplacer la suppression d’un produit dans la liste).

Rendu final



Mes Courses Zones Produits

Liste des Produits

[+ Nouveau Produit](#)

Zone	Nom	actions
	spaghetti (<i>Lustucuit</i>) - 500g Bon rapport qualité prix	✎ 🗑
Entrée du magasin	Croquettes pour chat	✎ 🗑
Conserves	Cassoulet (<i>Beauduel</i>) - 500g	✎ 🗑
Conserves	Haricots Verts (<i>Beauduel</i>) - 250g	✎ 🗑
Fruits et Légumes	Oignons	✎ 🗑
Fruits et Légumes	Pommes	✎ 🗑

Le code du fichier `show.html.twig` a été modifié comme ceci :

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

{% block title %}Produits{% endblock %}

{% block stylesheets %}
    <style>
        .table td.fit,
        .table th.fit {
            white-space: nowrap;
            width: 1%;
        }
    </style>
{% endblock %}

{% block main_content %}
    <h1>Liste des Produits</h1>

    <p class="my-4">
        <a class="btn btn-sm btn-primary" href="{{ path('produit_new') }}"
            title="Nouveau produit"><i class="bi bi-plus-lg me-1"></i>Nouveau Produit</a>
    </p>

    <table class="table">
        <thead>
            <tr>
                <th class="text-center">Zone</th>
                <th>Nom</th>
                <th class="text-center fit">actions</th>
            </tr>
        </thead>
        <tbody>
            {% for produit in produits %}
                <tr>
                    <td class="text-center fit text-primary">
                        {% if produit.zone %}{ { produit.zone.nom }}{% endif %}
                    </td>
                    <td>
                        <h5>
                            {{ produit.nom }}
                            {% if produit.marque %}<em>({{ produit.marque }})</em>{% endif %}
                            {% if produit.quantite %}
                                <small class="text-info"> - {{ produit.quantite }}</small>
                            {% endif %}
                        </h5>
                        {% if produit.commentaires %}
                            <small class="text-muted">{{ produit.commentaires }}</small>
                        {% endif %}
                    </td>
                    <td class="text-center fit">
                        <a class="btn btn-link fs-5 p-0 mx-3" title="Editer le produit"
                            href="{{ path('produit_edit', {'id': produit.id}) }}"
                            <i class="bi bi-pencil-square"></i>
                        </a>
                        {{ include('produit/_delete_form.html.twig') }}
                    </td>
                </tr>
            {% else %}
                <tr>
                    <td colspan="3">Aucun produit enregistré</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>

{% endblock %}
```

Tri de la liste des produits par zone

Pour que la liste soit triée par *zone*, et par *nom de produit*, nous avons rajouté une fonction `findAllOrdered()` dans le repository `ProduitRepository` :

```
/**
 * @return Produit[] Returns an array of Produit objects
 */
public function findAllOrdered(): array {
    return $this->createQueryBuilder('p')
        ->leftJoin('p.zone', 'z')
        ->orderBy('z.position', 'ASC')
        ->addOrderBy('p.nom', 'ASC')
        ->getQuery()
        ->getResult()
    ;
}
```

Et nous l'avons utilisée dans le contrôleur à la place du `findAll()` :

```
#[Route('/', name: 'produit_index', methods: ['GET'])]
public function index(ProduitRepository $produitRepository): Response
{
    return $this->render('produit/index.html.twig', [
        'produits' => $produitRepository->findAllOrdered(),
    ]);
}
```

*Ce premier TP Symfony s'arrête là.
Que pouvons-nous envisager pour la suite ?*

- Créer une entité *Liste* (pour liste de courses)
- Créer une entité *Article* (produit et sa quantité ajouté à la liste de courses)
- Pages d'affichage/création/modification/suppression
- Page « courses en cours » avec du javascript pour cocher au fur et à mesure la mise dans le caddy des articles.

