



Symfony

Formation pour développer des applications web
structurées et performantes

Se faire connaissance entre formateur & apprenants

Halim OULAD MANSOUR



Diplômé d'un Master MIAGE



Développeur Fullstack (Symfony, JAVA, JS)



<https://ho-consulting.github.io/web-site-for-teaching/>



Introduction

Sprint 1

Sprint 2

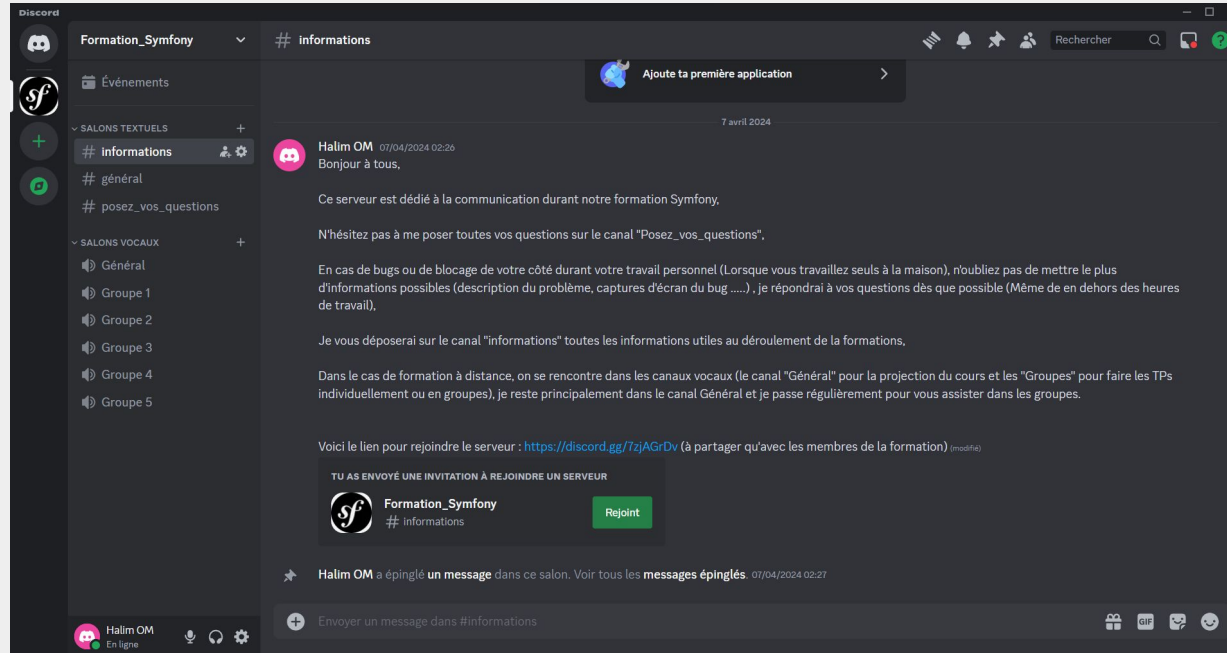
Sprint 3

Sprint 4

Sprint 5

Conclusion

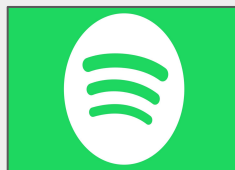
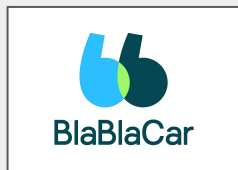
Afin d'avoir une communication permanente



Présentation

- Symfony est un framework MVC écrit en PHP.
- Des fonctionnalités modulables et adaptables.
- Permet l'accélération et la facilité du développement des applications web.

Ils ont choisi Symfony



- Liste complète des plateformes :
<https://my-flow.fr/quels-sont-les-sites-internet-qui-utilisent-symfony/frameworks-php/symfony/>

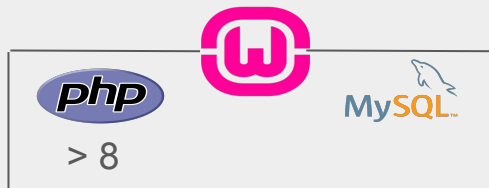
Objectif de la formation

- Comprendre les concepts de **l'architecture MVC**.
- Appréhender le développement et **les outils du framework Symfony**.
- Comprendre **l'ORM Doctrine** et faire le lien entre une application web Symfony et une base de données relationnelles.
- être capable de **développer** une application web **From Scratch** avec le framework **Symfony**.

Prérequis

- Des connaissances de bases en langages PHP / HTML / CSS.
- Des connaissances en bases de données.
- La programmation orienté objet.

Technologies requises et annexes



git



Déroulement de la formation

**Résultat****Résultat****Résultat****Résultat****Résultat**

Projet
Symfony en
place prêt au
développemen
t

La couche
métier de
l'application : les
entités,
contrôleurs
(avec les
fonctions CRUD)
et les vues.

Base de données
créée et
connectée avec
l'application,
possibilité
d'exécuter les
fonctionnalités
CRUD.

Les formulaires
sont ajoutés à
l'application,
possibilité
d'ajouter des
objets
directement via le
navigateur web.

Authentification
fonctionnelle,
l'utilisateur peut
se-connecter /
déconnecter et
exécuter que les
fonctions
appropriés à lui.

Plan de la formation et aspects abordés

- **Introduction** : Préparation de l'environnement et la mise en place du framework (En local).
- **Sprint 1** : Mise en place d'un projet symfony et découverte de sa structure.
- **Sprint 2** : L'architecture MVC en Symfony (Modèles, Vues, Contrôleurs)
- **Sprint 3** : L'ORM et les bases de données.
- **Sprint 4** : Les formulaires.
- **Sprint 5** : La sécurité (authentification et autorisation).
- **Conclusion** : Résultats et accomplissement.
- **Aller plus loin ?** : Mettre en place un environnement Javascript pour le front.

Préparation de l'environnement et la mise en place du framework (En local)

1- Télécharger et installer **WampServer** pour avoir un serveur **Apache2**, **PHP** et serveur de base de données **MySQL** en local.

Lien : <https://www.wampserver.com/>

Une fois c'est fait, démarrez les service Wamp et tester la présence de **PHP** et **Mysql** dans votre système en tapant sur le terminal les commandes suivantes :

```
php --version / mysql --version
```



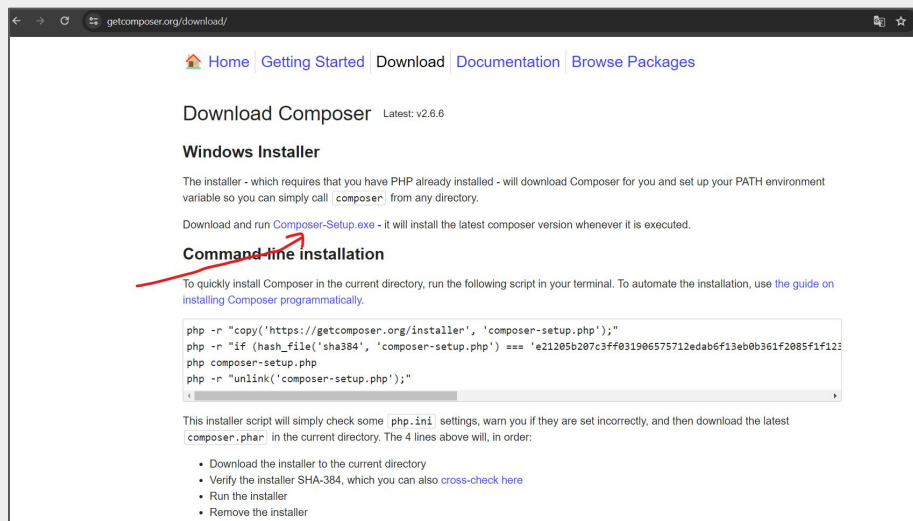
Préparation de l'environnement et la mise en place du framework (En local)

2- Télécharger & Installer **Composer**, c'est le gestionnaire de dépendances qui nous permettra d'installer des bibliothèques et les dépendances dont le projet aura besoin.

lien : <https://getcomposer.org/download/>

Une fois cela est fait, allez tester la présence de **composer** en tapant la commande suivante :

```
composer --version
```



Préparation de l'environnement et la mise en place du framework (En local)

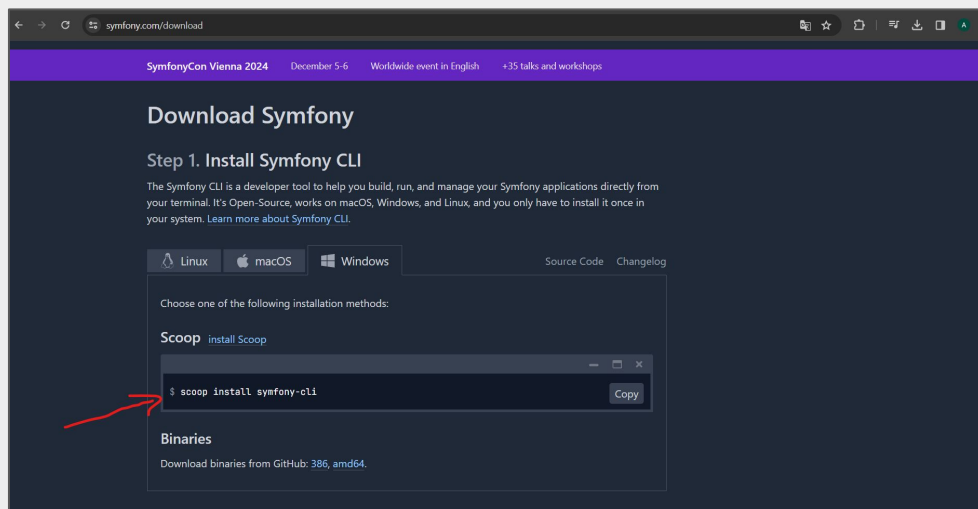
3 - Installer le **client Symfony** afin de créer / gérer nos application web Symfony.

Ouvrez le terminal **Windows PowerShell** et tapez la commande suivante :

```
scoop install symfony-cli
```

Allez confirmer si **symfony-cli** a été bien installé en tapant la commande suivante :

```
symfony --version
```

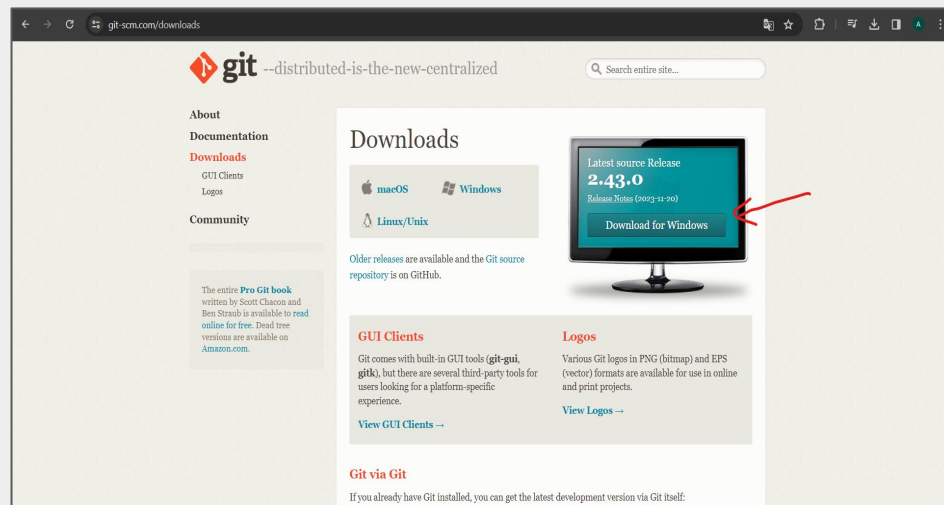


Préparation de l'environnement et la mise en place du framework (En local)

4- Télécharger et installer **Git**, c'est l'outil qui nous permettra de versionner l'évolution de notre application et l'avancement dans le code.

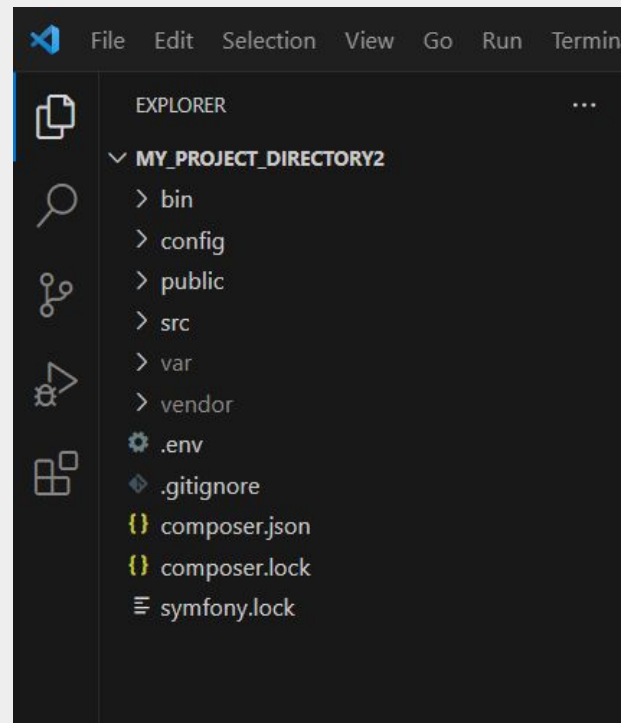
Lien : <https://git-scm.com/downloads>

Une fois cela est fait, confirmer l'installation en tapant la commande suivante : `git --version`



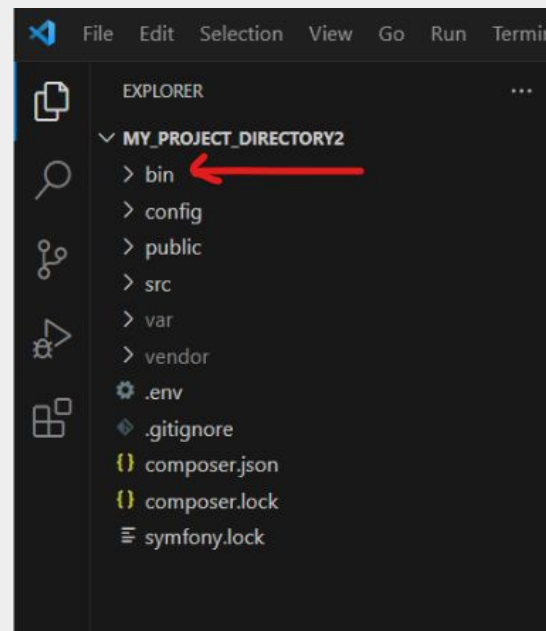
Découverte de la structure d'un projet Symfony

Voici la structure de base d'un projet symfony, ce sont ces fichiers auxquels on s'intéresse, on va les décortiquer un par un.



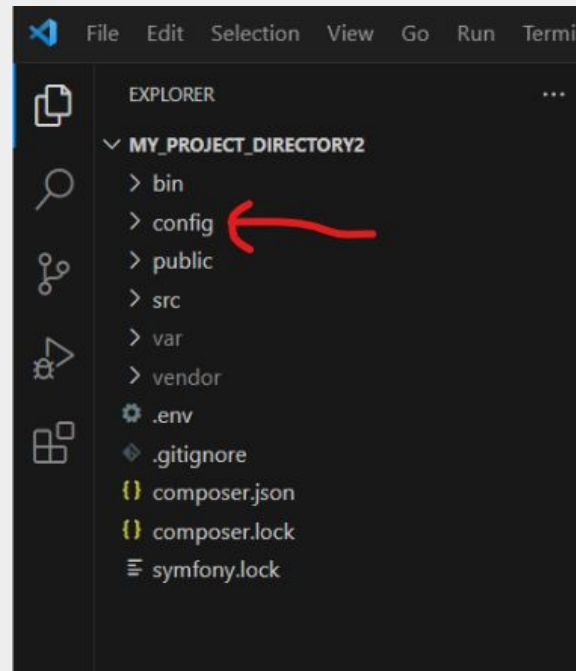
Découverte de la structure d'un projet Symfony

Le dossier dossier **bin** pour les binaries, il contient les fichiers qui nous sert à exécuter les commande de symfony-cli



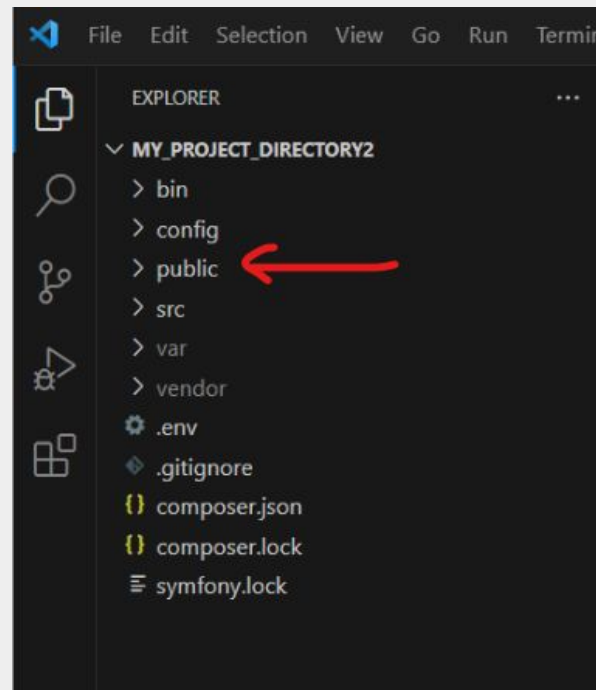
Découverte de la structure d'un projet Symfony

Le dossier config comme son nom l'indique, contient la configuration de notre application Symfony



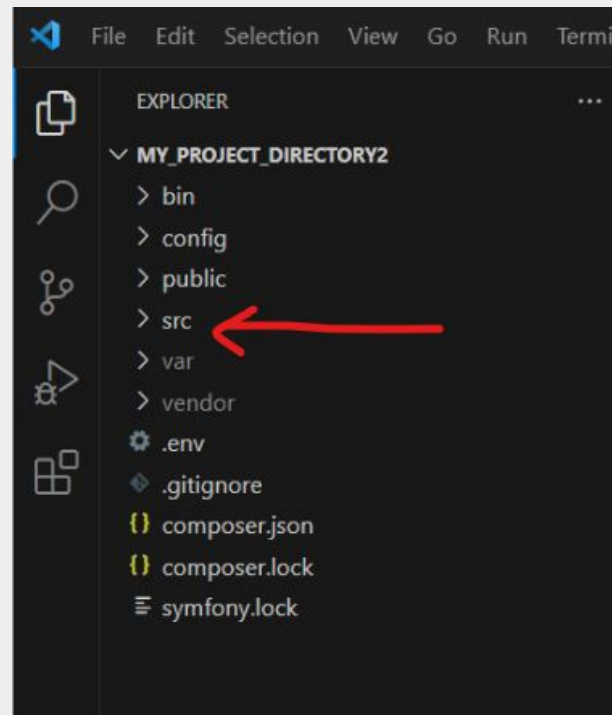
Découverte de la structure d'un projet Symfony

Le dossier public, comme son nom l'indique aussi, contiendra tous les fichiers publics de l'application comme les fichiers statiques.



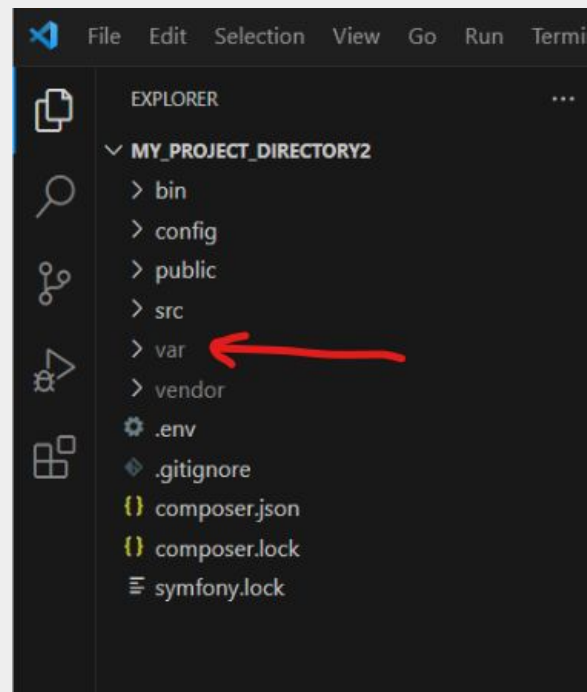
Découverte de la structure d'un projet Symfony

Le dossier **src**, c'est le dossier dans lequel on passe la majorité du temps à coder, src est l'abréviation du mot source, et comme son nom l'indique aussi, tout notre code source côté serveur se trouvera dans ce dossier.



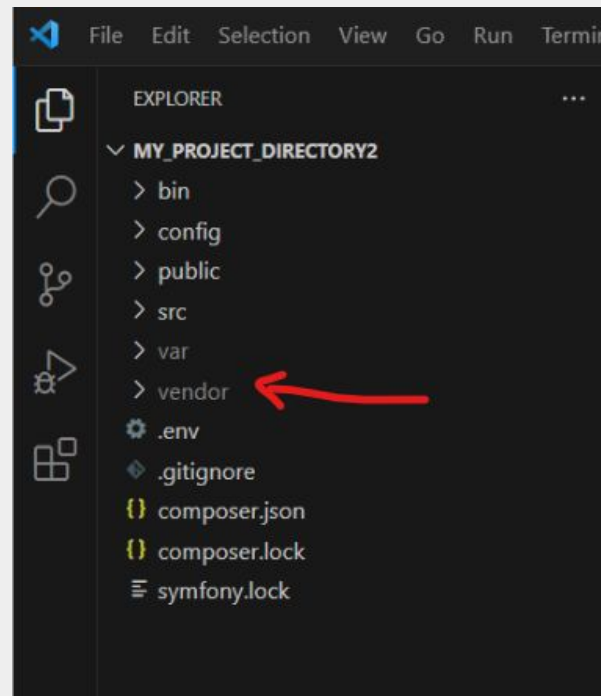
Découverte de la structure d'un projet Symfony

Le dossier var qui contiendra nos fichiers temporaires comme le cache les logsetc



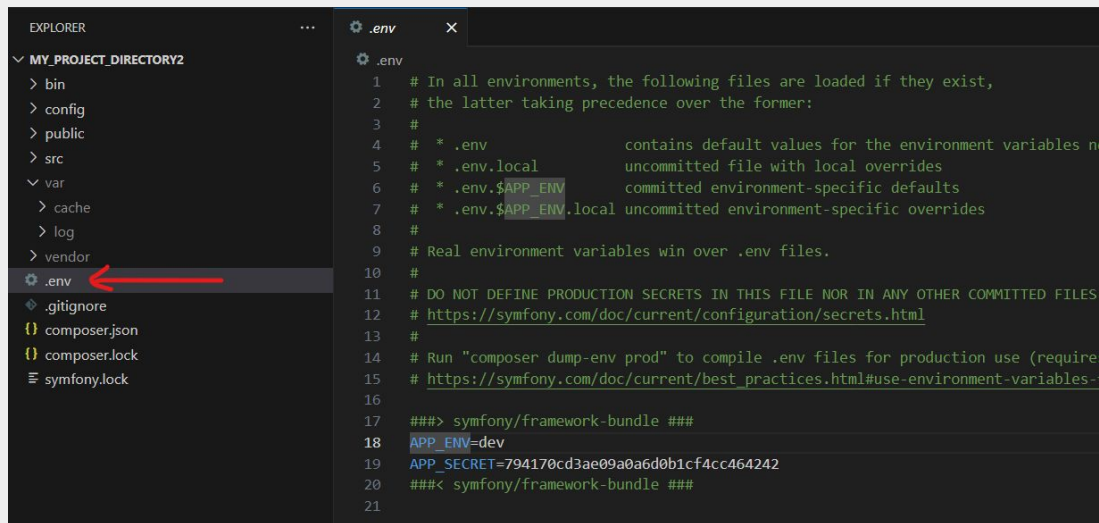
Découverte de la structure d'un projet Symfony

lorsque ces dépendances sont installés, ils vont se retrouver sur le dossier vendor



Découverte de la structure d'un projet Symfony

le fichier **.env** contiendra nos **variables d'environnement**, il nous sert à définir et faire la liaison entre notre applications et les outils qu'on doit utiliser (base de données, nos APIs externes ...) toutes les chaînes de connexions avec l'extérieur de Symfony seront définies dans ce fichier.

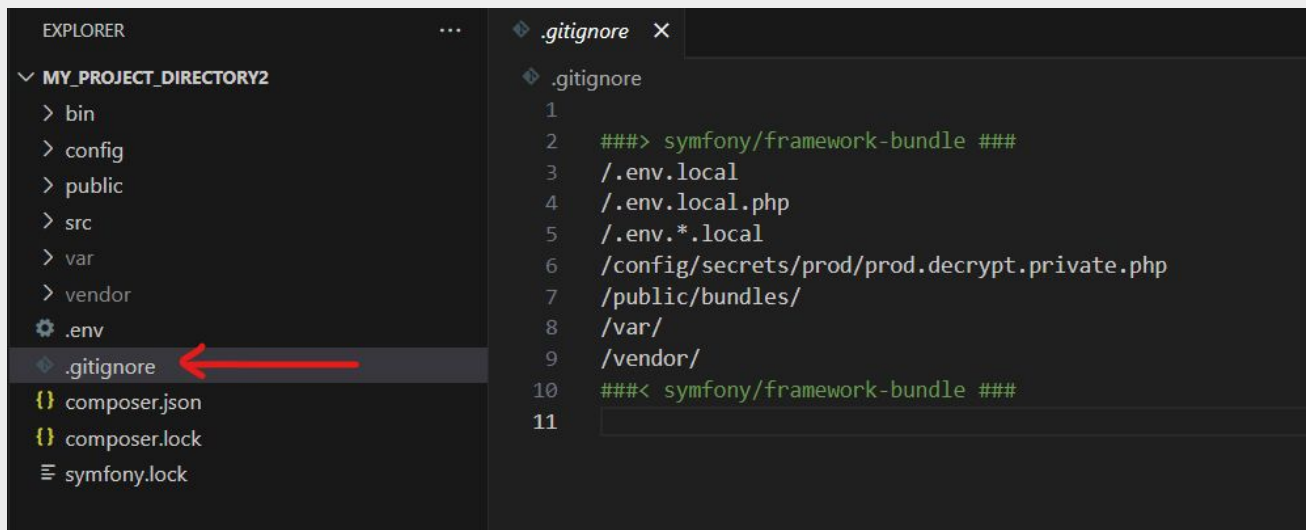


```
EXPLORER
MY_PROJECT_DIRECTORY2
  bin
  config
  public
  src
  var
    cache
    log
    vendor
  .env
  .gitignore
  composer.json
  composer.lock
  symfony.lock

.env
1 # In all environments, the following files are loaded if they exist,
2 # the latter taking precedence over the former:
3 #
4 # * .env contains default values for the environment variables n
5 # * .env.local uncommitted file with local overrides
6 # * .env.$APP_ENV committed environment-specific defaults
7 # * .env.$APP_ENV.local uncommitted environment-specific overrides
8 #
9 # Real environment variables win over .env files.
10 #
11 # DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILES
12 # https://symfony.com/doc/current/configuration/secrets.html
13 #
14 # Run "composer dump-env prod" to compile .env files for production use (require
15 # https://symfony.com/doc/current/best_practices.html#use-environment-variables-
16
17 ###> symfony/framework-bundle ###
18 APP_ENV=dev
19 APP_SECRET=794170cd3ae09a0a6d0b1cf4cc464242
20 ###< symfony/framework-bundle ###
21
```

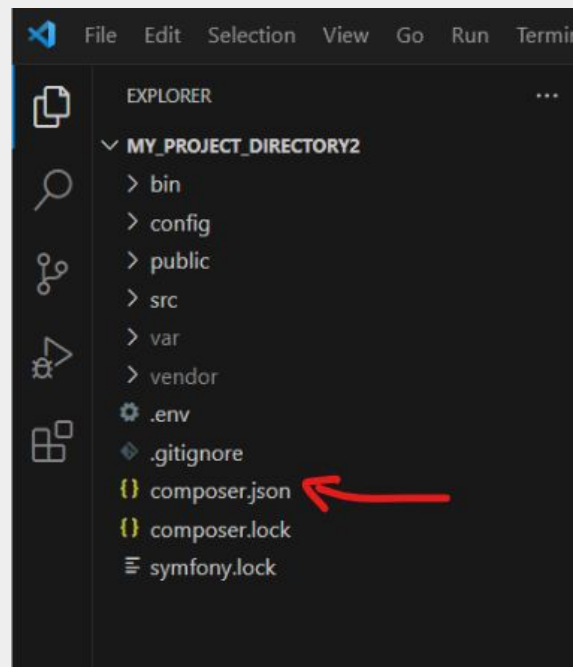
Découverte de la structure d'un projet Symfony

Le fichier **.gitignore** contient la liste des fichiers qui ne seront pas pris en comptes par Git.



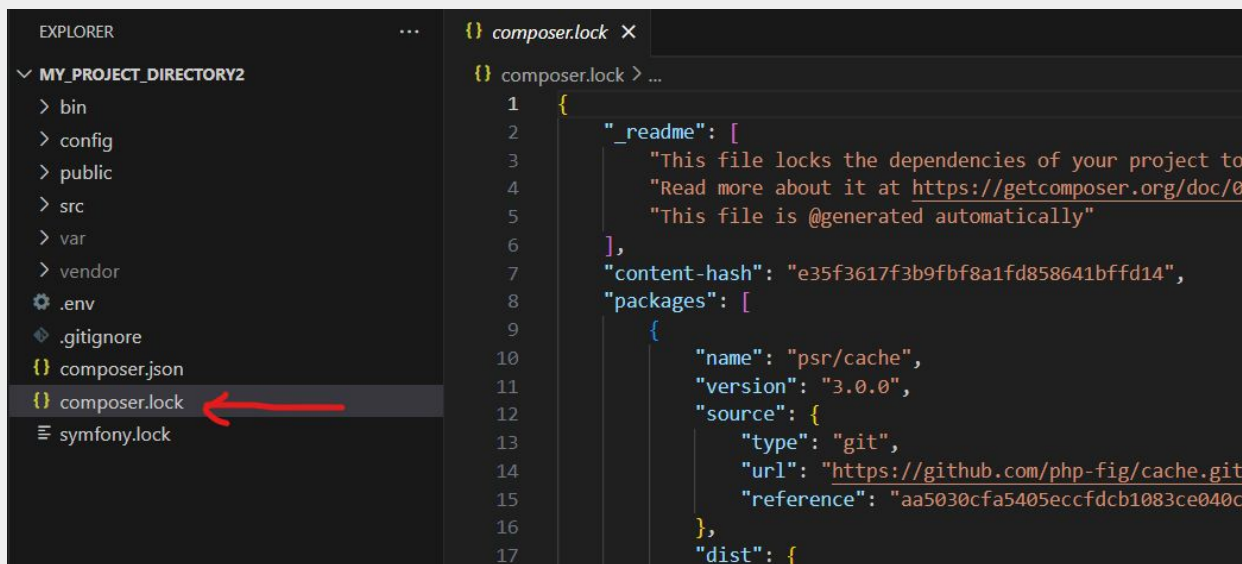
Découverte de la structure d'un projet Symfony

Le fichier `composer.json` est le fichier qui nous permettra de gérer tous les packages liés à notre projet, il contient la liste de toutes les dépendances du projet



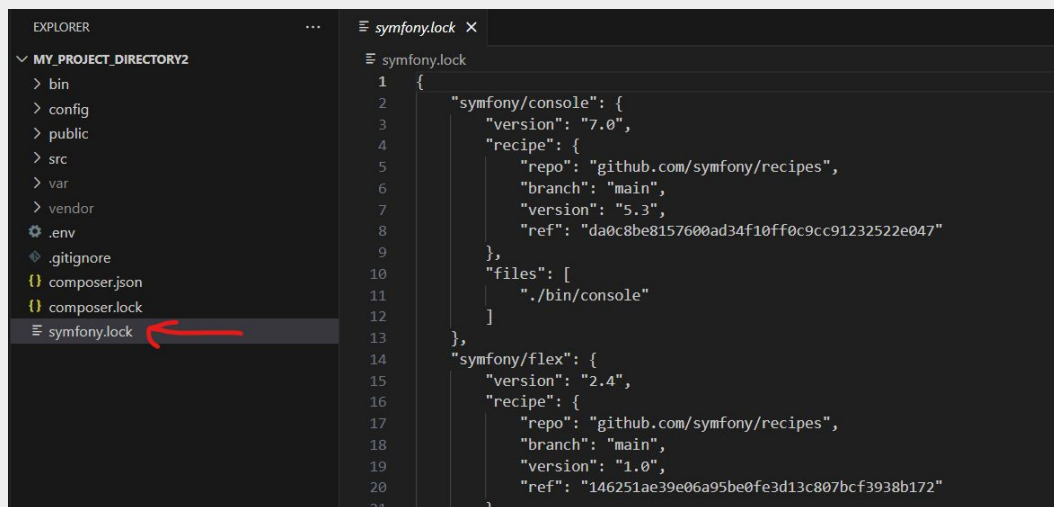
Découverte de la structure d'un projet Symfony

Le fichier **composer.lock** pour fixer les versions des dépendances installées, ceci permet de récupérer les mêmes versions en cas réinstaller le projet dans un autre système.



Découverte de la structure d'un projet Symfony

Le fichier **symfony.lock** est le fichier de config de **symfony-flex** qui nous permettra d'ajouter/supprimer des bundles et faciliter leurs gestion.

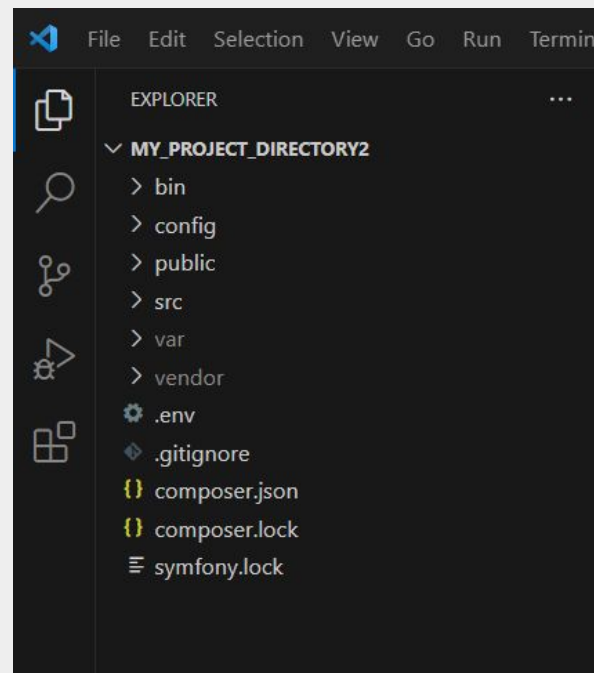


Découverte de la structure d'un projet Symfony

Ceci est la **structure** basique d'un projet Symfony, durant le développement de nos TPs, on sera amené à rajouter des bundles et des packages spécifique à nos besoins,

Pour ajouter un bundle à notre projet, on utilisera la commande :
`composer require la_reference_du_bundle`

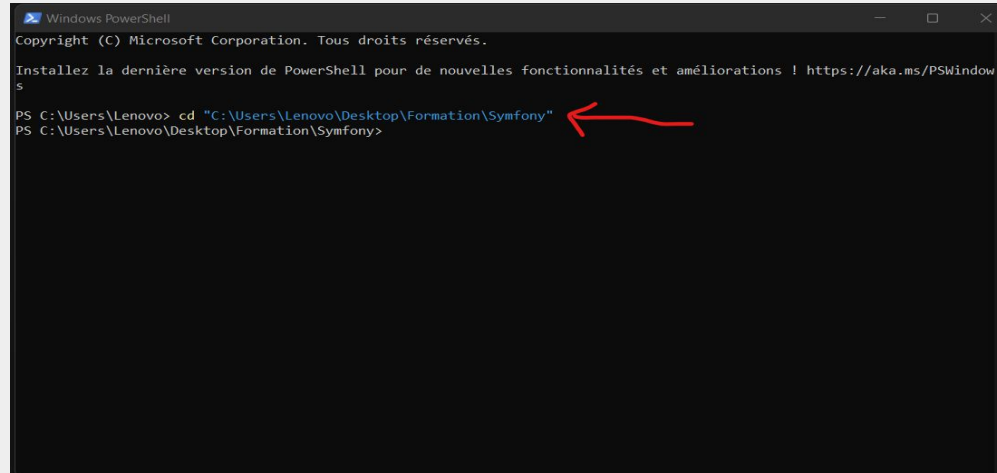
Pour supprimer un bundle, ça sera la commande :
`composer rem la_reference_du_bundle`



Mise en place d'un projet Symfony

Afin de créer un projet Symfony :

- On se positionne via le terminal dans l'emplacement souhaité (dans notre exemple c'est le dossier **"C:\Users\Lenovo\Desktop\Formation\Symfony"**) utilisant la commande **cd**.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

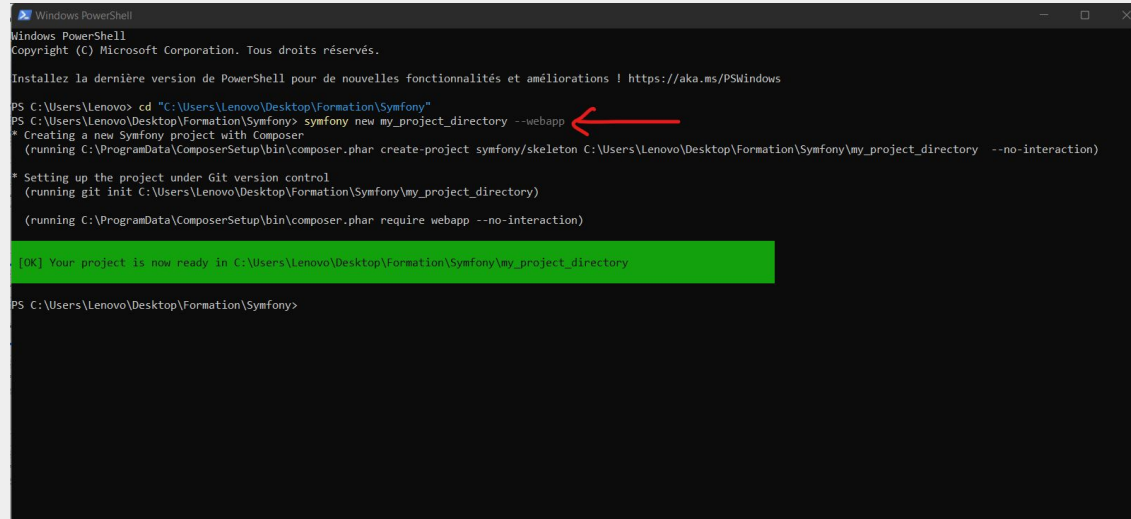
Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations ! https://aka.ms/PSWindows

PS C:\Users\Lenovo> cd "C:\Users\Lenovo\Desktop\Formation\Symfony"
PS C:\Users\Lenovo\Desktop\Formation\Symfony>
```

A red arrow points to the command `cd "C:\Users\Lenovo\Desktop\Formation\Symfony"` in the PowerShell terminal.

Mise en place d'un projet Symfony

- Une fois c'est fait, on tape la commande suivante :
`symfony new my_project_directory --webapp`



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations ! https://aka.ms/PSWindows

PS C:\Users\Lenovo> cd "C:\Users\Lenovo\Desktop\Formation\Symfony"
PS C:\Users\Lenovo\Desktop\Formation\Symfony> symfony new my_project_directory --webapp
* Creating a new Symfony project with Composer
(running C:\ProgramData\ComposerSetup\bin\composer.phar create-project symfony/skeleton C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory --no-interaction)

* Setting up the project under Git version control.
(running git init C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory)

(running C:\ProgramData\ComposerSetup\bin\composer.phar require webapp --no-interaction)

[OK] Your project is now ready in C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory

PS C:\Users\Lenovo\Desktop\Formation\Symfony>
```

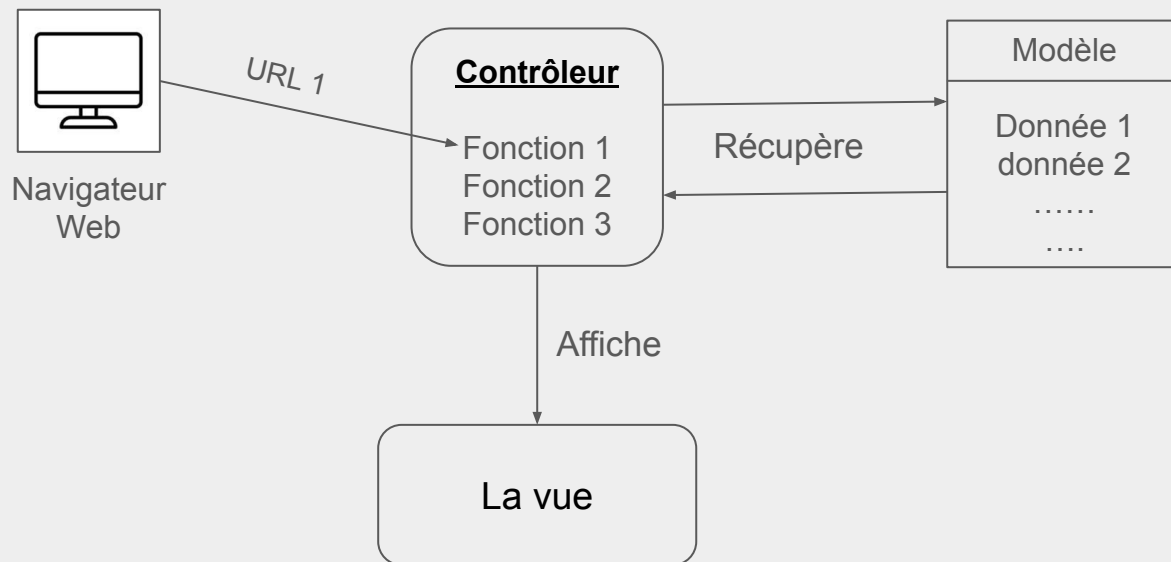
L'architecture MVC en Symfony

Que veut dire MVC ?

- **M => Modèle (Entité)** => représente la partie données (chaque entité est une table dans la bdd).
- **V => Vue (Template)** => représente l'interface de l'application (les écrans).
- **C => Contrôleur** => représente la partie traitement et désigne le comportement de l'application.

L'architecture MVC en Symfony

Comment ça se passe concrètement ?



L'architecture MVC en Symfony (Vues.twig)

Twig est un moteur de template intégré au framework Symfony avec une syntaxe sécurisée et simple à maîtriser.

Elle permet de conditionner l'affichage et maîtriser les données reçues dans la vue.

Syntaxe :

```
1  {% for user in users %}  
2      * {{ user.name }}  
3  {% else %}  
4      No users have been found.  
5  {% endfor %}
```

Boucler sur tableau de users

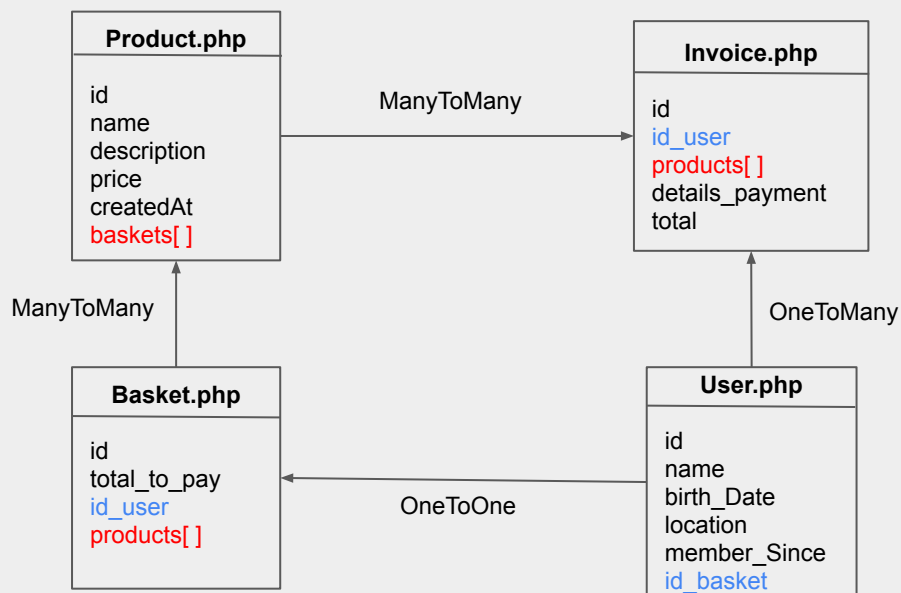
```
1  {% extends "layout.html" %}  
2  
3  {% block content %}  
4      Content of the page...  
5  {% endblock %}
```

Bloc d'instruction

L'architecture MVC en Symfony (Modèles)

Une entité (ou Modèle) en Symfony est une **classe.php** contient des attributs, elle peut être en relation avec d'autres entités via différents types de relation (OneToOne, OneToMany, ManyToOne, ManyToMany).

Exemple :



L'ORM (Object Relational Mapper) : Doctrine pour Symfony

L'ORM est le programme qui se place entre une application et sa base de données relationnelle associée, Le but c'est de pouvoir interagir avec la base de données à travers l'application (Création table / Interaction avec les données).

Chaque entité dans l'application représente une table au niveau de la base de données.

Côté Symfony

Product.php
id name description price createdAt

Mapping

`Doctrine(chaine_de_connexion_dans_fichier .env)`

Côté base de données

Product
id name description price created_at

L'ORM et les bases de données

- Pratiquement, le **Mapping des modèles vers tables** se fait en deux étapes :

Création de la migration : la création du code SQL en se basant sur les modèles (et les relations).

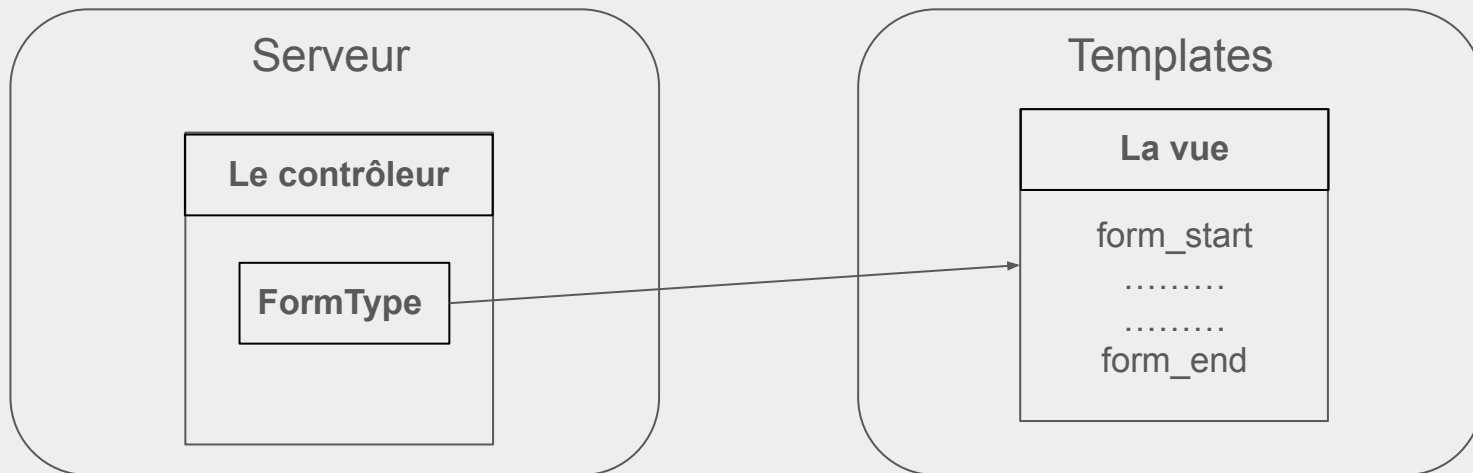
- Ceci se fait via la commande : `symfony console make:migration`
- Une fois cette commande est exécutée, on aura un **dossier migration** qui se crée et un fichier contenant les codes SQL à exécuter au niveau de la base de données.

L'exécution de cette migration afin de créer les tables sur la base de données.

- Ceci se fait via la commande : `symfony console doctrine:migrations:migrate`
- Après l'exécution de cette commande, les tables et les relations seront créées dans la base de données.

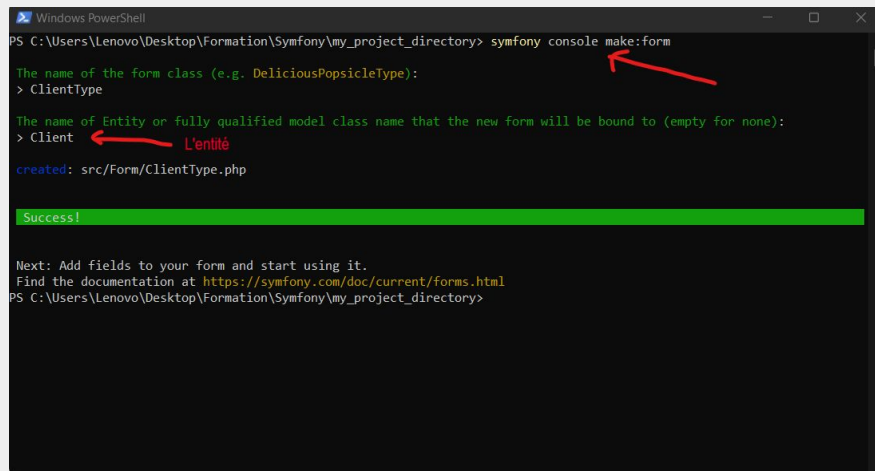
Les formulaires

- La mise en place d'un formulaire en Symfony d'une manière optimisée et sécurisée se fait via la logique suivante :



Les formulaires

- Tout d'abord, choisir l'entité autour de laquelle on souhaite créer un formulaire.
- Ensuite, créer le formulaire via la commande `symfony console make:form`



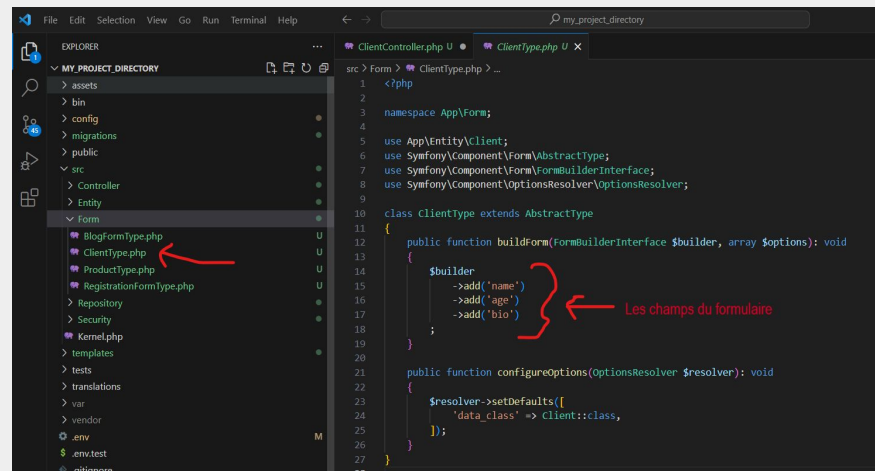
```
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory> symfony console make:form

The name of the form class (e.g. DeliciousPopsicleType):
> ClientType

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
> Client
created: src/Form/ClientType.php

Success!

Next: Add fields to your form and start using it.
Find the documentation at https://symfony.com/doc/current/forms.html
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory>
```



```
ClientController.php U • ClientType.php U X
src > Form > ClientType.php > ...
1 <?php
2
3 namespace App\Form;
4
5 use App\Entity\Client;
6 use Symfony\Component\Form\AbstractType;
7 use Symfony\Component\Form\FormBuilderInterface;
8 use Symfony\Component\OptionsResolver\OptionsResolver;
9
10 class ClientType extends AbstractType
11 {
12     public function buildForm(FormBuilderInterface $builder, array $options): void
13     {
14         $builder
15             ->add('name')
16             ->add('age')
17             ->add('bio')
18         ;
19     }
20
21     public function configureOptions(OptionsResolver $resolver): void
22     {
23         $resolver->setDefaults([
24             'data_class' => Client::class,
25         ]);
26     }
27
28 }
```

Les formulaires

- Tout d'abord, choisir l'entité autour de laquelle on souhaite créer un formulaire.
- Ensuite dans le contrôleur, créer l'action (la méthode) dans laquelle on veut instancier le formulaire et l'envoyer à la vue associée (généralement la méthode **new** / **edit**) qui servent à la création et modification des objets.

```
ClientController.php U X
src > Controller > ClientController.php > ClientController > new
15 class ClientController extends AbstractController
25 #[Route('/new', name: 'app_client_new', methods: ['GET', 'POST'])]
26 public function new(Request $request, EntityManagerInterface $entityManager): Response
27 {
28     $client = new Client();
29     $form = $this->createForm(ClientType::class, $client);
30     $form->handleRequest($request);
31
32     if ($form->isSubmitted() && $form->isValid()) {
33         $entityManager->persist($client);
34         $entityManager->flush();
35
36         return $this->redirectToRoute('app_client_index', [], Response::HTTP_SEE_OTHER);
37     }
38
39     return $this->render('client/new.html.twig', [
40         'client' => $client,
41         'form' => $form,
42     ]);
43 }
```

1 Instanciation du formulaire précédemment créé.

3 Récupération des informations du formulaire et enregistrer l'objet en BDD

Envoi du formulaire instancié à la vue

Les formulaires

- La dernière étape, recevoir le formulaire dans la vue afin de l'afficher.
- plusieurs fonction twig permettent une grande maîtrise de la mise en forme du formulaire qui soient :

`{{ form }}` : permet d'afficher tout le formulaire

`{{ form_start }}` : permet de générer la balise `<form>` avec les différents attributs

`{{ form_end }}` : permet de générer la fermeture de `<form>` avec les différents champs restants non affichés

`{{ form_errors }}` : affiche les erreurs éventuelles du formulaire

`{{ form_widget(mon formulaire.nomduchamps) }}` : affiche le type de champs



```
new.html.twig X
templates > client > new.html.twig
1  {% extends 'base.html.twig' %}
2
3  {% block title %}New Client{% endblock %}
4
5  {% block body %}
6      <h1>Create new Client</h1>
7
8      {{ form_start(form) }}
9          {{ form_widget(form) }}
10         <button class="btn">{{ button_label|default('Save') }}</button>
11     {{ form_end(form) }}
12
13
14     <a href="{{ path('app_client_index') }}">back to list</a>
15 {% endblock %}
16
```

Affichage du formulaire dans la vue

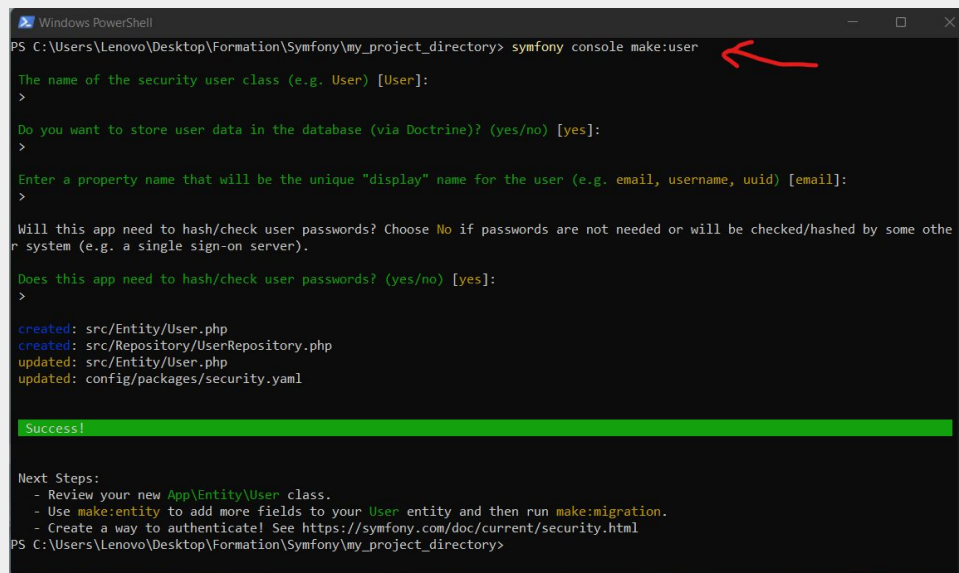
Sécurité

- L'aspect de la sécurité se divise sur deux partie :
 - Authentification (Qui êtes vous ?) : L'application Symfony identifie l'utilisateur.
 - Autorisation (Vous faites quoi ?) : L'application Symfony gère le comportement de cet utilisateur.

Sécurité

- Avant tout, ajoutons le composant nécessaire au fonctionnement de la sécurité via la commande : `composer require symfony/security-bundle` (Pas nécessaire si notre projet est une webapp)
- Création de l'entité User via la commande : `symfony console make:user`,

Ici Symfony va poser plusieurs questions et proposer des réponses par défaut (nom de l'entité = user, vouloir stocker les infos en bdd = yes, l'identifiant unique pour la cnx = email, encodage du mot de passe = yes) : Laissez toutes les réponses par défauts.



```
Windows PowerShell
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory> symfony console make:user

The name of the security user class (e.g. User) [User]:
>

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
>

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
>

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).
Does this app need to hash/check user passwords? (yes/no) [yes]:
>

created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml

Success!

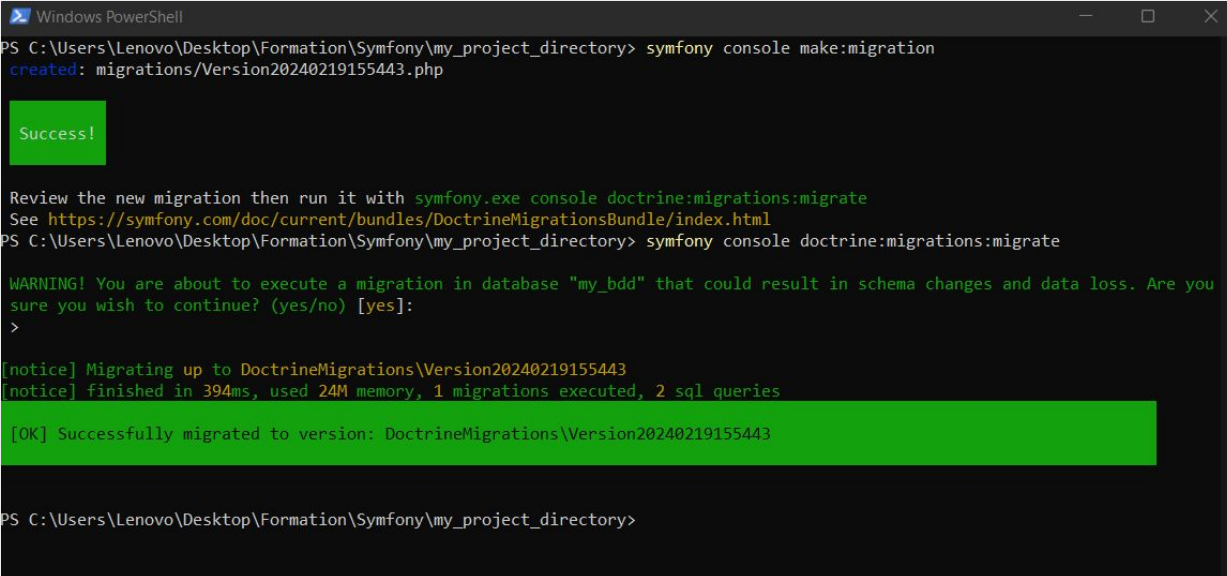
Next Steps:
- Review your new App\Entity\User class.
- Use make:entity to add more fields to your User entity and then run make:migration.
- Create a way to authenticate! See https://symfony.com/doc/current/security.html
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory>
```

Sécurité

- Analyse des fichiers créés / modifiés :
 - **L'entité User (le provider)**: Une simple entité avec certaines propriétés qui lui permettent d'être utilisée pour l'authentification dans l'application Symfony (comme l'attribut role).
 - **UserRepository** : Le repository propre à l'entité.
 - Le fichier de configuration **security.yaml** : Ce fichier fait le lien avec le provider (l'user), le login retenu (par défaut "l'email") et l'encodage du mot de passe (par défaut "auto").

Sécurité

- Mise à jour de la bdd avec la nouvelle table user via les commandes :
`symfony console make:migration` `symfony console doctrine:migrations:migrate`



```
Windows PowerShell
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory> symfony console make:migration
created: migrations/Version20240219155443.php

Success!

Review the new migration then run it with symfony.exe console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory> symfony console doctrine:migrations:migrate

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

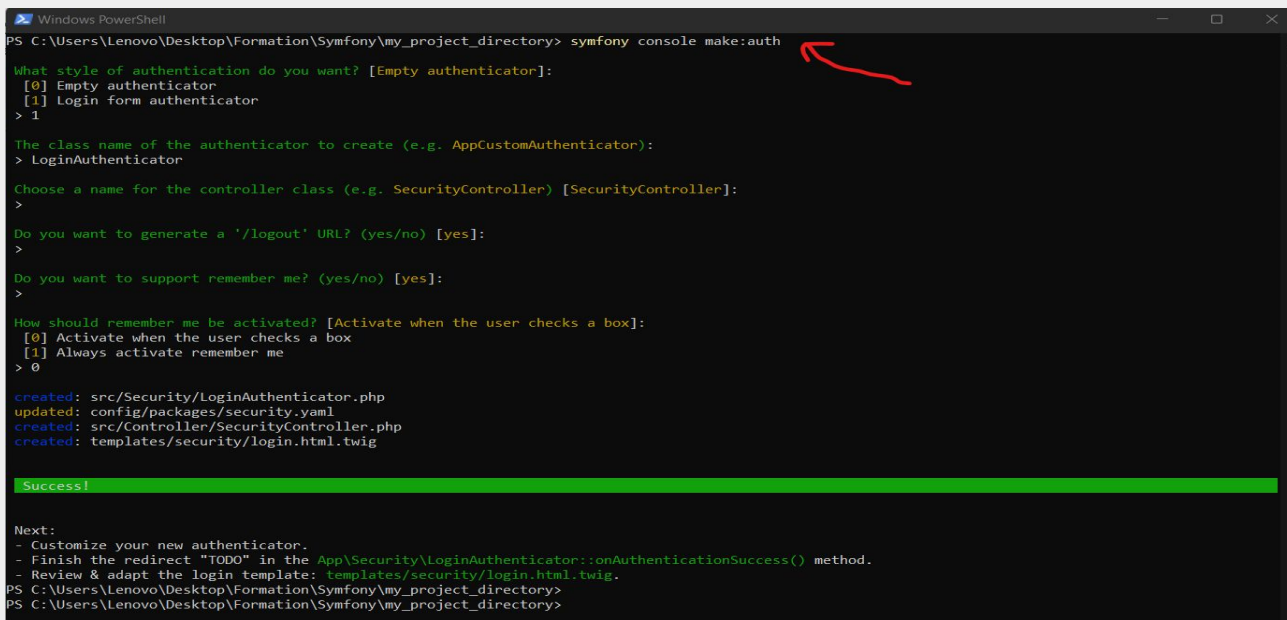
[notice] Migrating up to DoctrineMigrations\Version20240219155443
[notice] finished in 394ms, used 24M memory, 1 migrations executed, 2 sql queries

[OK] Successfully migrated to version: DoctrineMigrations\Version20240219155443

PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory>
```

Sécurité

- Création du système de la partie connexion via la commande : `symfony console make:auth`



```
Windows PowerShell
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory> symfony console make:auth

What style of authentication do you want? [Empty authenticator]:
  [0] Empty authenticator
  [1] Login form authenticator
> 1

The class name of the authenticator to create (e.g. AppCustomAuthenticator):
> LoginAuthenticator

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
>

Do you want to generate a '/logout' URL? (yes/no) [yes]:
>

Do you want to support remember me? (yes/no) [yes]:
>

How should remember me be activated? [Activate when the user checks a box]:
  [0] Activate when the user checks a box
  [1] Always activate remember me
> 0

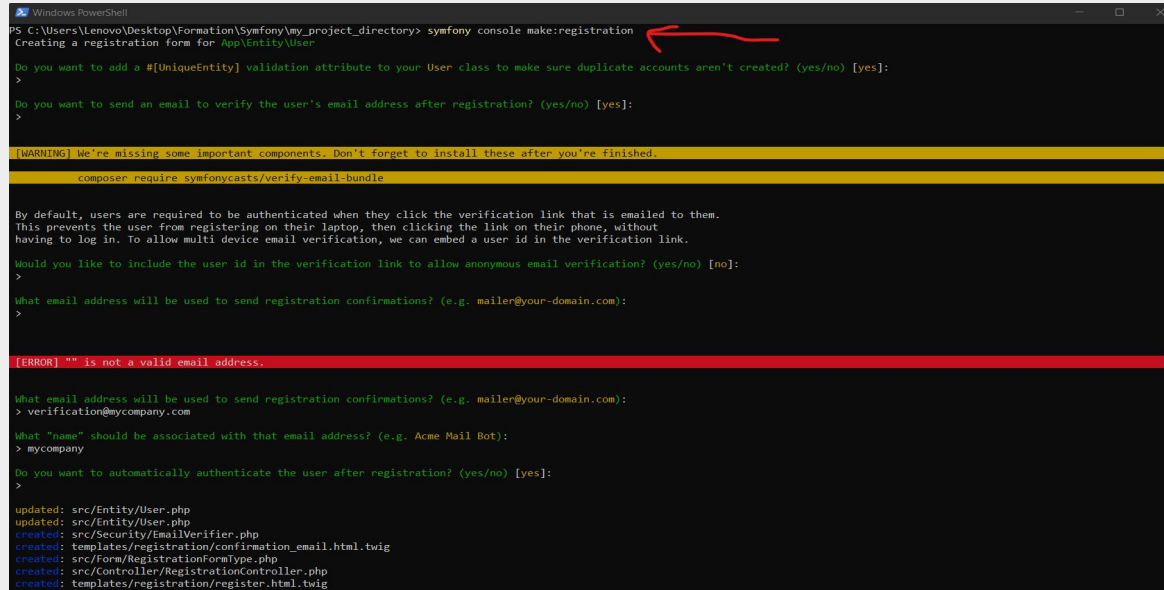
created: src/Security/LoginAuthenticator.php
updated: config/packages/security.yaml
created: src/Controller/SecurityController.php
created: templates/security/login.html.twig

Success!

Next:
- Customize your new authenticator.
- Finish the redirect "TODO" in the App\Security\LoginAuthenticator::onAuthenticationSuccess() method.
- Review & adapt the login template: templates/security/login.html.twig
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory>
```

Sécurité

- Création du système de l'inscription via la commande : `symfony console make:registration`



```
PS C:\Users\Lenovo\Desktop\Formation\Symfony\my_project_directory> symfony console make:registration
Creating a registration form for App\Entity\User

Do you want to add a #[UniqueEntity] validation attribute to your User class to make sure duplicate accounts aren't created? (yes/no) [yes]:
>

Do you want to send an email to verify the user's email address after registration? (yes/no) [yes]:
>

[WARNING] We're missing some important components. Don't forget to install these after you're finished.

composer require symfonycasts/verify-email-bundle

By default, users are required to be authenticated when they click the verification link that is emailed to them.
This prevents the user from registering on their laptop, then clicking the link on their phone, without
having to log in. To allow multi device email verification, we can embed a user id in the verification link.

Would you like to include the user id in the verification link to allow anonymous email verification? (yes/no) [no]:
>

What email address will be used to send registration confirmations? (e.g. mailer@your-domain.com):
>

[ERROR] "" is not a valid email address.

What email address will be used to send registration confirmations? (e.g. mailer@your-domain.com):
> verification@mycompany.com

What "name" should be associated with that email address? (e.g. Acme Mail Bot):
> mycompany

Do you want to automatically authenticate the user after registration? (yes/no) [yes]:
>

updated: src/Entity/User.php
updated: src/Entity/User.php
created: src/Security/EmailVerifier.php
created: templates/registration/confirmation_email.html.twig
created: src/Form/RegistrationFormType.php
created: src/Controller/RegistrationController.php
created: templates/registration/register.html.twig
```

Sécurité

- Analyse des fichiers créés / modifiés par la commande `symfony console make:registration` :
 - **RegistrationController** : C'est le contrôleur qui gère le mécanisme d'inscription dans une application Symfony ainsi que la validation de l'email.
 - **RegistrationFormType** : Le formulaire d'inscription des utilisateurs de l'application Symfony.
 - **register.html.twig** : La vue qui permet l'inscription
 - **User** : La mise à jour de l'entité User (Dans le cas d'ajout de la confirmation email).

Aller plus loin ?

- Mettre en place environnement NodeJS pour le front.
- Moderniser et dynamiser notre front (Javascript, CSS, Bootstrap).



Compilés



Injectés



Interprétés

