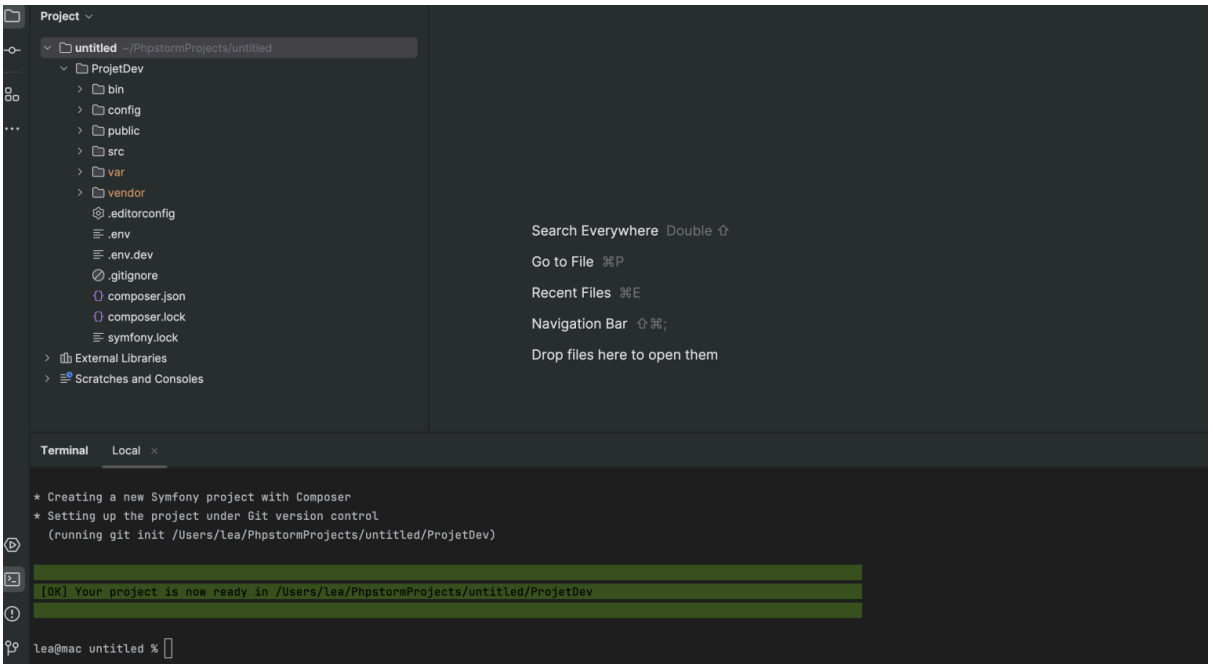
Commandes importantes de Symfony

Pré-requis :

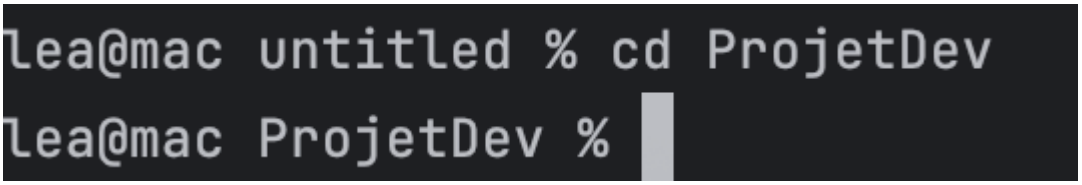
- Symfony d'installé avec un serveur de base de données, php et serveur web (comme apache)
- un IDE (Phpstorm,VScode etc.)

1. Créer un projet Symfony

Créer un projet dans le répertoire souhaité avec la commande dans un terminal "symfony new". Symfony va créer un dossier avec de nombreux fichiers pour commencer à développer



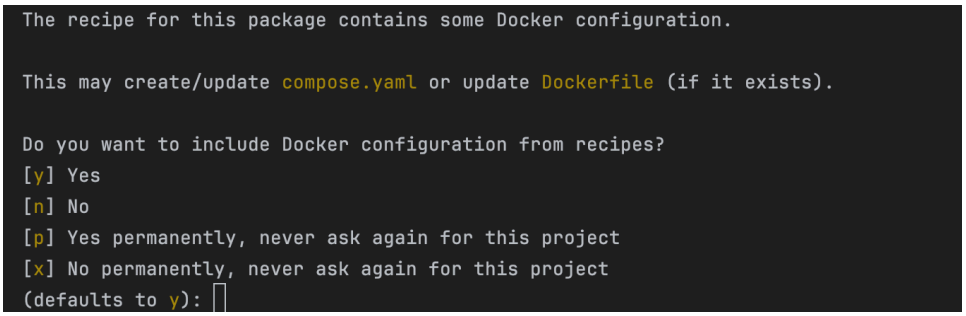
Ne pas oublier de se rendre dans le chemin du dossier dans son terminal (terminal de serveur ou terminal interne à l'IDE) avec la commande : "cd *nom_du_projet"*



2. Plusieurs dépendances à ajouter

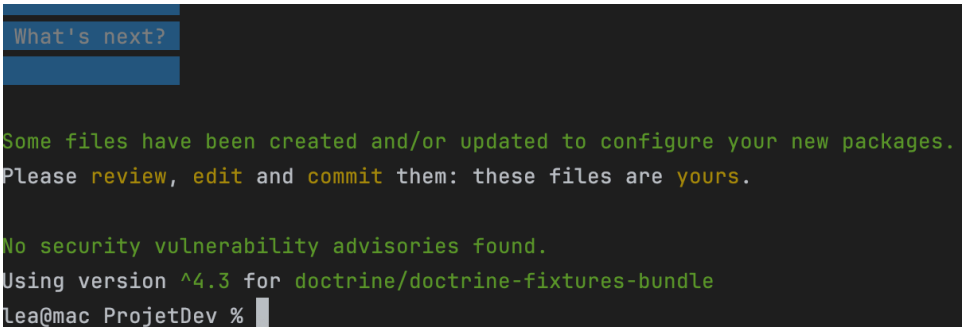
Plusieurs dépendances sont nécessaires à la création d'un projet Symfony, installés avec les commandes "composer require" :

- **Doctrine** (composer require doctrine). Elle fournit l'ORM (interface) permettant de gérer la base de données à partir des entités

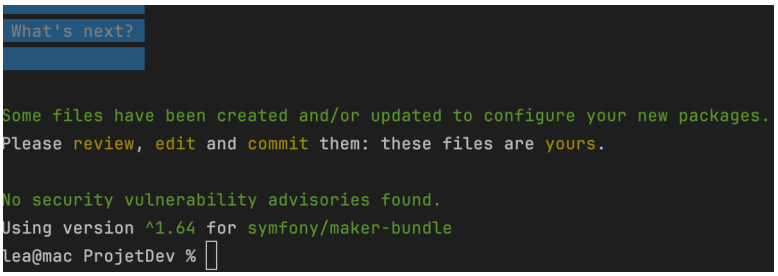


Dans le cas où nous n'utilisons pas Docker pour un projet, il n'est pas nécessaire d'ajouter cette configuration. On peut alors répondre "n" ou "x".

- **Fixtures** (composer require doctrine/doctrine-fixtures-bundle). Elle permet de remplir la base de données avec des données de tests.



- **Twig** (composer require twig). Elle intègre le moteur de templates twig pour créer des vues dynamiques.
- **Maker Bundle** (composer require maker –dev). Elle facilite la génération de code (entités, formulaires, controllers, CRUD) grâce à ses commandes "make:" en environnement de développement.



- **Form** (composer require form). Elle permet de créer, d'afficher et de gérer les formulaires d'une application.
- **Security** (composer require security). Elle ajoute les fonctionnalités d'authentification,la gestion d'utilisateurs, mots de passe et ses rôles, l'accès aux routes.

- **Validator** (composer require validator). Elle permet de valider les données enregistrées dans un formulaire
- **CSRF** (composer require symfony/security-csrf). Elle protège les formulaires grâce à une génération de tokens, jetons

3. Connecter son projet à sa BDD

Un fichier est à modifier : **.env** à la racine du projet créé par Symfony, 2 possibilités :

- Créer un fichier **.env.local** copié du fichier .env pour développer (recommandé pour séparer la configuration de la partie en développement et d'un projet en production)
- Modifier le fichier.env directement

En ayant bien installé Doctrine, dans le **.env.local**, commenter les lignes qui ne concernent pas la base de données utilisée et décommenter (avec le #) la ligne avec la base de données utilisée (mysql, mariadb, postgresql).
Il faut alors changer la ligne dans cette syntaxe :
DATABASE_URL="mysql://utilisateur:mot_de_passe@127.0.0.1:3306/nom_de_la_base"
Remplacer les valeurs **utilisateur**, **mot_de_passe**, et **nom_de_la_base**.

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/co
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data_%kernel.environment%.db"
# DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8.0.32&charset=utf8mb4"
DATABASE_URL="mysql://user:password@127.0.0.1:8080/BaseDev"
# DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=16&charset=utf8"
###< doctrine/doctrine-bundle ###
```

Après avoir configuré l'URL de la base de données, créer la base avec :
php bin/console doctrine:database:create

```
lea@mac ProjetDev % php bin/console doctrine:database:create
Created database `BaseDev` for connection named default
lea@mac ProjetDev %
```

4. Démarrer son serveur Symfony

Pour démarrer son serveur : symfony server:start

```
[Web Server ] Nov  8 14:48:11 |DEBUG  | PHP    Reloading PHP versions
[Web Server ] Nov  8 14:48:11 |DEBUG  | PHP    Using PHP version 8.4.4 (from default version in $PATH)
[Web Server ] Nov  8 14:48:11 |INFO   | PHP    listening path="/opt/homebrew/Cellar/php/8.4.4/sbin/php-fpm" php="8.4.4" port=49772
[Web Server ] Nov  8 14:48:11 |DEBUG  | PHP    Using PHP version 8.4.4 (from default version in $PATH)
[PHP-FPM    ] Nov  8 14:48:11 |NOTICE | FPM    fpm is running, pid 5468
[PHP-FPM    ] Nov  8 14:48:11 |NOTICE | FPM    ready to handle connections
[PHP-FPM    ] ls (first boot)"}
[PHP-FPM    ] Nov  8 14:48:13 |DEBUG  | RUNNER Received timer message (first boot) cmd="PHP-FPM"
```

(Pour le lancer en arrière-plan : symfony serve -d)

```
[OK] Web server listening
     The Web server is using PHP FPM 8.4.4
     http://127.0.0.1:8000


Stream the logs via symfony server:log
Lea@mac ProjetDev %
```

Pour éteindre son serveur : symfony server:stop ou Ctrl + C

```
[PHP-FPM    ] ls (first boot)"}
[PHP-FPM    ] Nov  8 14:48:13 |DEBUG  | RUNNER Received timer message (first boot) cmd="PHP-FPM"
^C
Shutting down! Waiting for all workers to be done.


 [OK] Stopped all processes successfully


lea@mac ProjetDev %
```

5. Créer une entité

Pour générer une entité on utilise la commande **php bin/console make:entity.** Un formulaire de plusieurs questions sera alors affiché. Ici, on va essayer de créer une entité “voiture” avec le modèle, nombre de places dans la voiture, l’année, comme champs.

- On renseigne le nom de l’entité.

```
lea@mac ProjetDev % php bin/console make:entity

 Class name of the entity to create or update (e.g. OrangeElephant):
 > Voiture
```

- On renseigne le nom du champ dans l’entité.
- On renseigne le type de l’entité (appuyer sur “?” permet de voir tous les types possibles).

```
New property name (press <return> to stop adding fields):
> modele

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

```
Main Types
  * string or ascii_string
  * text
  * boolean
  * integer or smallint or bigint
  * float

Relationships/Associations
  * relation a wizard 🧙 will help you build the relation
  * ManyToOne
```

- Ici “modele” est un string. On peut appuyer sur entrée pour continuer

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


Field length [255]:
>
```

- On renseigne la longueur du champs (255).
- On renseigne si le champ peut être “null” dans la base de données.

```
Can this field be null in the database (nullable) (yes/no) [no]:
>
```

- On renseigne les autres champs (nbPlaces comme int, annee comme date)

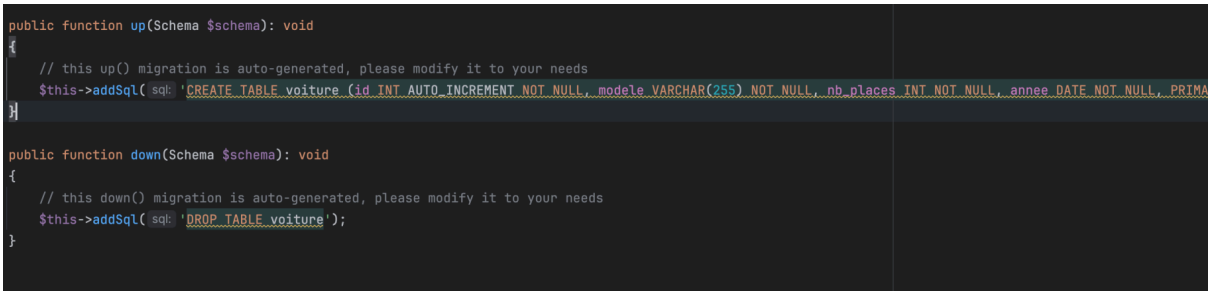
```
Add another property? Enter the property name (or press <return> to stop adding fields):
>
```

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

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

Can this field be null in the database (nullable) (yes/no) [no]:
>
```

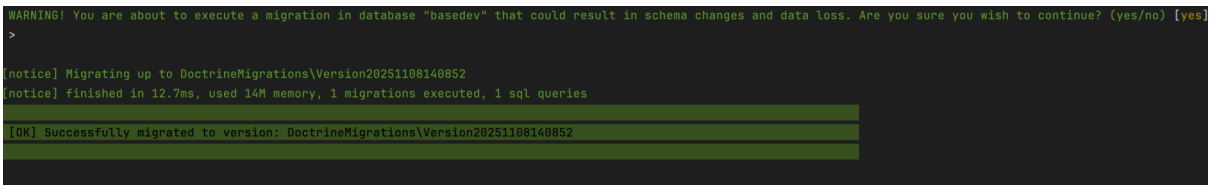
```
New property name (press <return> to stop adding fields):
> annee

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

Can this field be null in the database (nullable) (yes/no) [no]:
>
```

- Quand les entités sont créées on génère un fichier de migration avec : php bin/console make:migration

Ce fichier, dans le dossier “migrations”, précise toutes les entités créées avec les détails

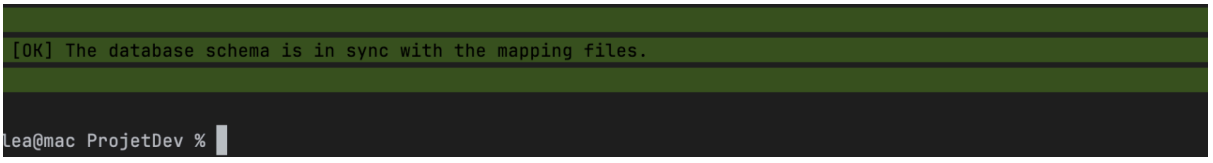


Si tout est en ordre, on applique la migration et les entités créées sont appliquées à la base de données avec la commande : **php bin/console doctrine:migrations:migrate**. Un message est affiché pour valider le choix d’appliquer les changements à la base de données. “Yes” est le choix par défaut.



On peut alors vérifier que la nouvelle entité existe bien sur notre base de données (exemple sur phpmyadmin)

On peut aussi s’assurer que le schéma de base de données soit bien à jour avec la commande : **php bin/console doctrine:schema:validate**



6. Générer un CRUD (Create, Read, Update, Delete)

On peut créer un CRUD grâce à la commande : **php bin/console make:crud NomDeLEntité**. Ici, *php bin/console make:crud Voiture*.
On demande le nom du Controller qui va gérer le CRUD. Ici VoitureController correspond parfaitement.

```
lea@mac ProjetDev % php bin/console make:crud Voiture

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

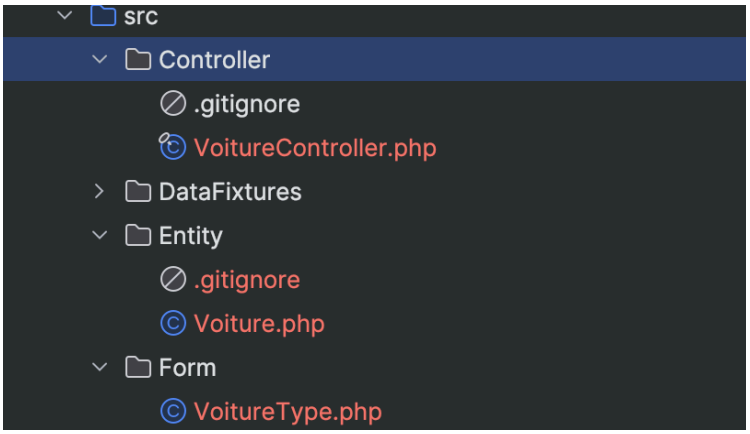
Les tests ne sont pas obligatoires.

```
Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]:
>
```

Le CRUD est créé.

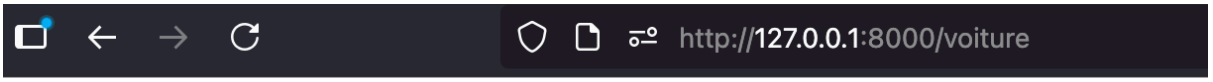
```
created: templates/voiture/index.html.twig
created: templates/voiture/new.html.twig
created: templates/voiture/show.html.twig


 Success!


Next: Check your new CRUD by going to /voiture/
ea@mac ProjetDev %
```

Plusieurs fichiers sont créés, un Controller pour gérer le contenu, le “Form” pour gérer le formulaire d’ajout



On peut vérifier le contenu dans le fichier “VoitureController” dans src/Controller/VoitureController.php et dans le navigateur à l’adresse. http://127.0.0.1:8000/voiture

```php
#[Route('/voiture')]
final class VoitureController extends AbstractController
{
    no usages
    #[Route(name: 'app_voiture_index', methods: ['GET'])]
    public function index(VoitureRepository $voitureRepository): Response
    {
        return $this->render( view: 'voiture/index.html.twig', [
            'voitures' => $voitureRepository->findAll(),
        ]);
    }

    #[Route('/new', name: 'app_voiture_new', methods: ['GET', 'POST'])]
    public function new(Request $request, EntityManagerInterface $entityManager): Response
    {
```



Voiture index

Id	Modele	NbPlaces	Annee	actions

no records found

Create new

On peut aussi créer un Controller manuellement avec la commande : **php bin/console make:controller NomDuController** (exemple : **php bin/console make:controller DashboardController** pour créer un controller de tableau de bord d’accueil.)