



# Symfony

## Projet 8 : Découverte du Framework PHP Symfony 5

### Table des matières

Projet 8 : Découverte du Framework PHP Symfony 5 .....	1
Ressources : .....	3
Sources : .....	3
Objectif : .....	4
Présentation : .....	4
4 Rôles : .....	4
A - Avant-Propos .....	5
B – Rappel protocole http.....	6
C - Requêtes et réponses en Symfony avec les composants HttpFoundation.....	7
SYMFONY C’EST QUOI ?.....	7
Un Framework pour quoi faire ? .....	8
AVANTAGES : .....	8
INCONVENIENT .....	9
D- Pourquoi Symfony ? .....	9

E – Le gestionnaire de paquets : Composer .....	9
F -Installer Symfony 5 : .....	10
G - Structure d'un projet de base .....	10
H - Introduction à Symfony Flex .....	13
SYMFONY UTILISE : .....	13
L'autowiring de services (auto câblage) .....	14
<a href="https://symfony.com/doc/5.4/components/yaml/yaml_format.html">https://symfony.com/doc/5.4/components/yaml/yaml_format.html</a> .....	14
Autowiring liste : .....	14
Le composant EventDispatcher en bref .....	15
I – Configurer une application .....	16
Symfony et MVC (Models Views Controllers) .....	16
J – Les Contrôleurs : Dossier src / Controller .....	17
LES CONTROLLERS.....	18
Les composants HttpFoundation : .....	20
K-L 'Objet Request : .....	20
L – L'Objet Response :.....	22
M - Les FlashBags : .....	24
N - LE SYSTEME DE ROUTE :.....	26
O- Les Vues : Dossier src / templates .....	28
P - TWIG un moteur de Template.....	31
La syntaxe : .....	32
R - La barre de debug Profiler.....	36
S - Symfony Flex dans le détail.....	36
T -La couche modèle avec Doctrine (ORM).....	37
U - DOCTRINE ORM ET DQL : .....	38
V – Le fichier de configuration .env et MySQL : .....	38
W – Créer une base de données : .....	39
X – Créer une entité (Table et / ou Classe) et des migrations (fichier script) : .....	39
Y – Entity/Produit.php .....	40
Les migrations .....	41

Z – Fixtures ou faux jeux de données .....	43
METHODE 1 : FAKER .....	44
Z-1 – Créer un contrôleur et afficher les données de la table produit.....	46
Z-2 - AFFICHER UNE PAGINATION : .....	50
Z-3 – LES DETAILS D’UN PRODUIT .....	52
Z-4 – SUPPRIMER UN PRODUIT .....	54
Z-5 – AJOUTER UN PRODUIT.....	55
Z-6 – EDITER UN PRODUITS .....	60
Z-7 – Le langage DQL (Doctrine Query Language) .....	61
Z-8 - UN SYSTEME D’INSCRIPTION ET DE CONNEXION SECURISÉ AVEC DES ROLES .....	64
Z-9 – Les relation entre les entités .....	71
Z-10 – UNE RELATION ONE TO ONE .....	71
Z-11 – Une relation Many To Many et les tables intermédiaires .....	75
Z-12 – Une relation One To Many et Many To One .....	77
Z-13 – Le Reverse Engineering : .....	80
Z-14 - ADMINISTRATION : LE BUNDLE EASY ADMIN .....	80
Z-15 – AJOUTER UN TABLEAU DE BORD POUR LES UTILISATEURS.....	86
Z-17 – AJOUTER UNE BARRE DE RECHERCHE PAR FOURCHETTE DE PRIX .....	91
Z-18 – LES ASSERTS ENTITE POUR SECURISE VOS TABLES .....	97

Ressources :

DEFINITION :

**Symfony** est un ensemble de composants PHP, ainsi qu’ un Framework MVC libre écrit avec le langage de programmation PHP. Il fournit des fonctionnalités modulables et adaptables qui permettent de faciliter et d’accélérer le développement d'un site web.

Sources :

Site officiel :

<https://symfony.com/doc/5.4/index.html>

<https://symfonycasts.com/tracks/symfony5>

Open Classroom :

<https://openclassrooms.com/fr/courses/5489656-construisez-un-site-web-a-l-aide-du-framework-symfony-5>

Divers :

[https://www.google.com/search?q=tuto+symfony+5&rlz=1C1ONGR\\_frFR978FR978&source=lnms&tbs=vid&sa=X&ved=2ahUKEwj8xaS3q7b4AhXXhM4BHUhDDuAQ\\_AUoAXoECAEQAw&biw=1920&bih=929&dpr=1](https://www.google.com/search?q=tuto+symfony+5&rlz=1C1ONGR_frFR978FR978&source=lnms&tbs=vid&sa=X&ved=2ahUKEwj8xaS3q7b4AhXXhM4BHUhDDuAQ_AUoAXoECAEQAw&biw=1920&bih=929&dpr=1)

Composer :

<https://getcomposer.org/>

Packagist:

<https://packagist.org/>

Twig:

<https://twig.symfony.com/>

**Objectif :**

Le but de ce projet est de créer une plateforme de vente de produit de base : avec le Framework Symfony version 5.4.

**Présentation :**

Dans ce projet, il est important de comprendre et de repartir les possibilités d'accès aux données en fonction des rôles des utilisateurs.

**4 Rôles :**

- SIMPLE VISITEUR
- ROLE\_USER
- ROLE\_ADMIN
- ROLE\_SUPER\_ADMIN

1 – Le visiteur :

- Consulter des produits
- Rechercher des produits

- Contacter les vendeurs
- Acheter des biens

## 2 – Le vendeur : ROLE\_USER

- Inscription et connexion
- Accès sécurisé au tableau de bord (BACKEND)
- Gestion des produits via un tableau de bord (CRUD)

## 3 – Administrateur : ROLE\_ADMIN

- Accès au Bundle EasyAdmin 3 pour manager la plateforme

## 4 – SUPER ADMIN : ROLE\_SUPERADMIN

- Créer des administrateurs, les éditer et les supprimer

## A - Avant-Propos :

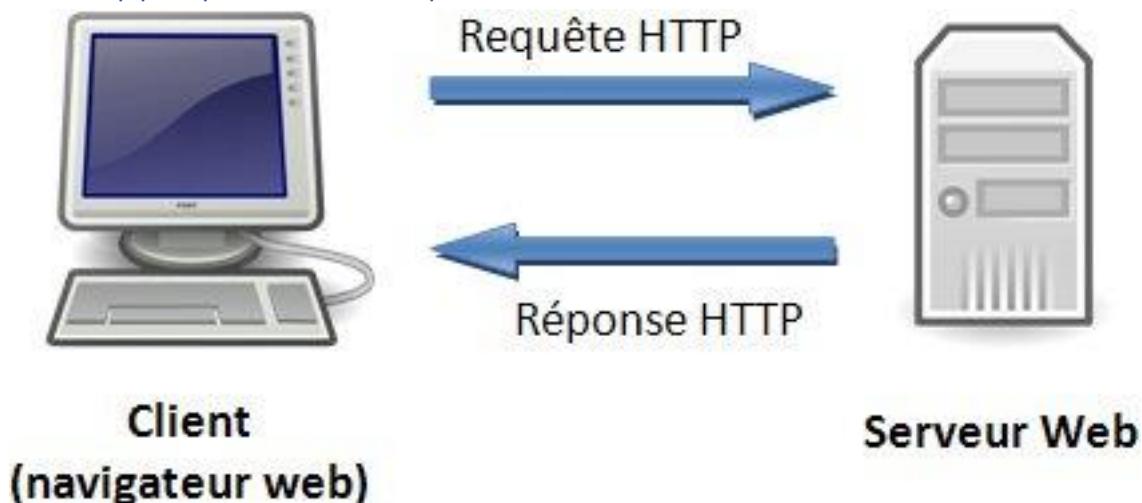
1. Composer
2. Installer Symfony
3. Structure de Symfony
4. Le routage
5. Le moteur de Template Twig
6. Le profiler Symfony
7. Symfony Flex
8. La couche Modèle Doctrine
9. Les formulaires
10. La sécurité
11. Les pages d'erreurs
12. La multi langue Internationalisation
13. Les Services
14. Swift Mailer (email)
15. Déployer son Site

## A - Avant-Propos

- 1- Être patient
- 2- Ne pas vouloir tous maîtriser (Symfony = assemblage de module complexe)

- 3- Savoir trouver l'information sur internet : (Taper les mots clés, communauté Symfony, vérifié la configuration du projet, version du Framework, date de la solution, etc...) Symfony.com, php.net, stackoverflow, tuto, coach, Symfony Cast, Packagist, etc...
- 4- Être organisé : MCD, Diagramme de séquence, etc...
- 5- Ne pas chercher à tous développer : 😞 Ne pas réinventer la roue, un bon IDE, étape par étape, ressources disponible)

## B – Rappel protocole http



Exemple :

- 1) Une url : https://exemple.com = nom de domaine = DNS (Domain Name Server)
- 2) Cette adresse permet de retrouver l'adresse IP du serveur qui l'héberge le site. (Exemple ; Wamp utilise une IP locale ; 127.0.0.1)
- 3) Chaque PC possède la même IP local : IP : 127.0.0.1 = ceci est comme numéro de téléphone
- 4) Le téléphone = Navigateur Web (Chrome, Firefox etc...)
- 5) Navigateur -> Nom de domaine -> connexion au serveur = (envoi requête http du navigateur au serveur) -> Réponse du serveur au Navigateur

6) La requête est traitée par le serveur et envoie une réponse

Les Codes réponses http (HyperText transfert Protocol)

[https://fr.wikipedia.org/wiki/Liste\\_des\\_codes\\_HTTP](https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP)

- Les codes 1xx, sont des informations
- Les codes 2xx sont des messages de succès
- Les codes 3XX, qui signalent les redirections de ressources et qu'il existe plusieurs réponses possibles
- Les codes 4XX, qui signalent une erreur de client http, de requête côté client (Votre PC) ;
- Les codes 5XX, qui signalent une erreur côté serveur.

## C - Requêtes et réponses en Symfony avec les composants HttpFoundation

Le Framework Symfony et notamment ses composants HttpFoundation, apporte une couche d'abstraction pour les requêtes et les réponses, il est simple à utiliser et à manipuler.

Tous accès à une application web se fait via une requête http (un entête (DNS, type de contenus, etc...) et un corps (body)).

Le serveur suite à la réception de la requête envoie une réponse : avec Symfony ces 2 éléments sont 2 composants (classes) Request + Response

- 2 Classe : Request et Response sont issues de HttpFoundation
- Pour les utilisés on doit importer les classes via auto loader : avec use
- Use Symfony\Component\HttpFoundation\Request
- Use Symfony\Component\HttpFoundation\Response

## SYMFONY C'EST QUOI ?

Symfony est un ensemble de composant lié par un noyau (le Kernel).

Le Framework Symfony est construit autour du **paradigme fondamental du web** : un utilisateur fait une requête et le serveur doit retourner une réponse.

- Le composant **HttpFoundation** fournit une couche d'abstraction PHP objet pour la requête et la réponse.
- Le composant **HttpKernel** a la responsabilité de récupérer la requête de l'utilisateur et de renvoyer une réponse.
- Le composant HttpFoundation définit une couche orientée objet pour la spécification HTTP.
- En PHP, la requête est représentée par des variables super globales (`$_GET`, `$_POST`, `$_FILES`, `$_COOKIE`, `$_SESSION`, ...) et la réponse est générée par des fonctions (`echo`, `header ()`, `setcookie ()`, ...).
- Le composant Symfony HttpFoundation remplace ces variables globales et fonctions PHP par défaut par une couche orientée objet.

Un **contrôleur Symfony** est une simple fonction (méthode) d'une classe PHP d'où il est possible de configurer le **routing** à l'aide d'annotations PHP même si d'autres formats de déclaration sont possibles.

Le Framework Symfony non seulement à un composant pour gérer le routing, mais fournit aussi un **contrôleur frontal** en charge de recevoir toutes les requêtes de l'utilisateur et de trouver la bonne action (fonction) du contrôleur à exécuter.

```
Routes = Annotation @Route("/accueil", name= "page_accueil")
```

## Un Framework pour quoi faire ?

### AVANTAGES :

- 1- Gain de temps (Classe et objet préconstruite, Autowiring de Service (auto-câblage de classe))
- 2- Un code déjà structuré
- 3- Un code standardisé pour un travail en groupe
- 4- Intégration et reprise de code d'un même Framework
- 5- Grosse communauté
- 6- Code, module, composant réutilisable
- 7- Les Framework évoluent (dernière version Symfony 6 + PHP 8)

## INCONVENIENT

- 1- Bibliothèques lourdes pas utilisées à 100% (Notamment avec Symfony version --full)
- 2- Fin du code personnalisé, tendances à trouver des solutions à l'aide du Framework ou de la communauté sur internet
- 3- Une période d'apprentissage en plus de la connaissance de PHP
- 4- Les Framework évoluent certaine version change beaucoup (ex : Symfony 2 à 3 avec la disparition des bundles + version les lignes de commande qui génère du code)

## D- Pourquoi Symfony ?

- 1- Grande communauté
- 2- Adaptabilité
- 3- Longévité (LTS : Long Time Support)
- 4- Framework libre de droit = Open Source

## E – Le gestionnaire de paquets : Composer

Composer est un logiciel gestionnaire de dépendances libre de droit écrit en PHP.

<https://getcomposer.org/>

Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin. Le développement a débuté en avril 2011 et a donné lieu à une première version sortie le 1<sup>er</sup> mars 2012.

Toutes les instructions sont réalisées en ligne de commande, dans un terminal (Rappel : cmd, gitbash ou terminal sous Windows)

Optionnel :

Initialiser composer vide (cette action va générer un fichier composer.json)

**composer init**

Installer un paquet :

**composer require nom du paquet**

Liste des paquets disponibles (PHP et autres Framework) :

<https://packagist.org/>

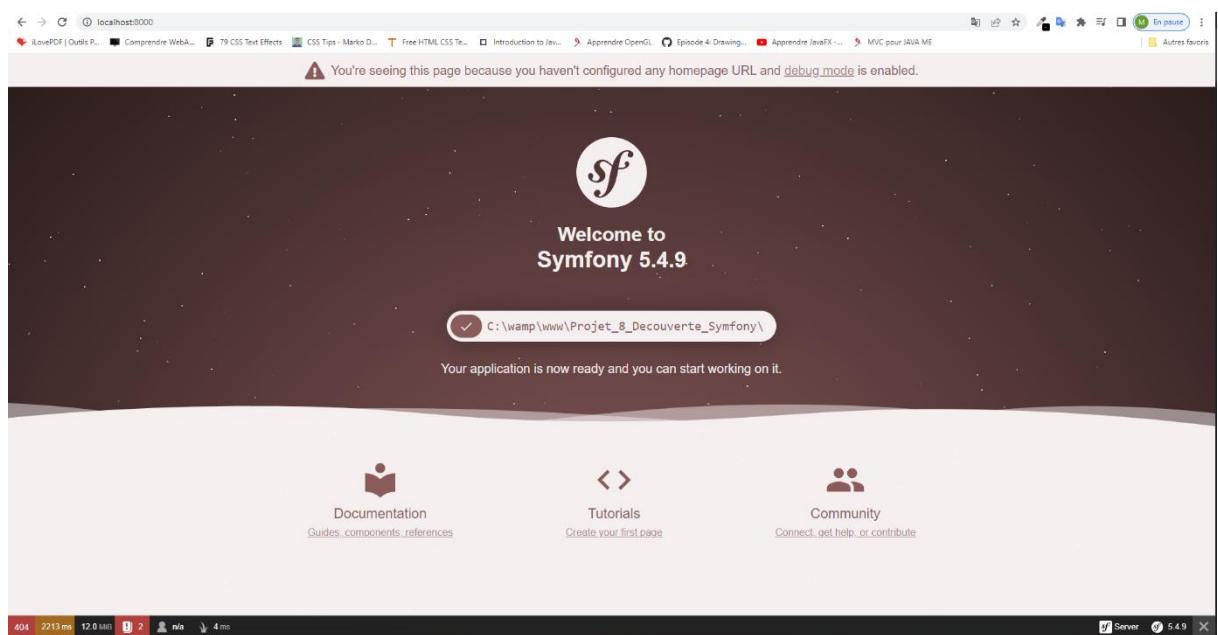
## F -Installer Symfony 5 :

*Par défaut nous allons utiliser l'installation de Symfony via composer*

- a- La Version minimal de PHP : 7.2.5
- b- <https://symfony.com/download> setup.exe
- c- Via composer : **composer create-project symfony/skeleton : "^5.4"**  
**nomDuProjet**

```
composer create-project symfony/skeleton:"^5.4" my_project_directory
```

- d- Lancer le server local Symfony : cmd -> **symfony server :start** ou **symfony serve** (**symfony server:stop** pour l'arrêter)
- e- URL (uniform resources locator) = **localhost :8000** ou **127.0.0.1 :8000**
- f- Pour changer de port : **symfony server :start –port 1234**



## G - Structure d'un projet de base

La structure de base Modèles Vues Contrôleurs (MVC) :

- Toutes les données sont dans le dossier src/
- Les modèles = src/Entity : Des données issues de votre base de données ou non, gérer via l'ORM DOCTRINE (Object Relational Mapping)
- Les vues = dossier templates/ : regroupent tous ce qui génère des pages HTML à retourner au client, c'est la partie visuelle de votre application.

Les vues sont des fichiers à l'extension .html.twig = Moteur ou langage de Template

- **Les contrôleurs** = src/Controller : regroupent tous les programmes PHP qui coordonne l'application, ils vont appeler les modèles et retourner les valeurs (données) aux vues, c'est donc le cœur de l'application.

## LA STRUCTURE D'UN PROJET

- **Le dossier "bin"**

Ce dossier contient les exécutables disponibles dans le projet, que ce soit ceux fournis avec le Framework (la console Symfony) ou ceux des dépendances (phpunit, simple-phpunit, php-cs-fixer, phpstan).

- **Le dossier "config"**

Il contient toute la configuration de votre application, que ce soit le Framework, les dépendances (Doctrine, Twig, Monolog) ou encore les routes (au format .yaml).

Ne pas oublier qu'il est possible d'adapter la configuration du Framework en fonction de l'environnement, et qu'une partie de la configuration se trouve aussi dans le fichier .env du projet.

- **Le dossier "public"**

Par défaut, il ne contient que le contrôleur frontal de votre application(index.php), le fichier dont la responsabilité est de recevoir toutes les requêtes des utilisateurs.

Seul ce dossier doit être accessible de l'extérieur.

C'est le seul dossier accessible par la requête client, il contient tous les fichiers pouvant être chargé par le navigateur (css, js, img, pdf, etc...)

- **Le dossier "migrations"**

Dans ce dossier et si vous manipulez une base de données, alors vous trouverez les migrations (requête DQL) de votre projet généré à chaque changement que vous effectuerez sur votre base de données à l'aide de l'ORM Doctrine.

- **Le dossier "src"**

C'est ici que se trouve le cœur votre application, les contrôleurs, formulaires, écouteurs d'événements, modèles (entité) et tous vos services doivent se trouver dans ce dossier. C'est également dans ce dossier que se trouve le "moteur" de votre application, le kernel.

- **Le dossier "tests"**

Dans ce dossier se trouvent les tests unitaires, d'intégration et d'interfaces.

Par défaut, l'espace de nom du dossier **tests** est App\Tests et celui du dossier **src** est App.

- **Le dossier "templates"**

Ce dossier contient les gabarits (les vues) qui sont utilisés dans votre projet, ce sont de fichier au format Twig (twig est un générateur de Template HTML)

- **Le dossier "translations"**

Symfony fournit un composant appelé Translation capable de gérer de nombreux formats de traductions, dont les formats yaml, xliff, po, mo... Ces fichiers seront situés dans ce dossier.

- **Le dossier "var"**

Ce dossier contient trois choses principalement :

- Les fichiers de cache dans le dossier **cache** ;
- Les fichiers de log dans le dossier **log** ;
- Et parfois, si le Framework est configuré pour gérer les sessions PHP dans le système de fichiers, on trouve le dossier **sessions**.

- Le dossier "vendor"

Ce dossier contient votre chargeur de dépendances (ou "autoloader") et l'ensemble des dépendances de votre projet PHP installées à l'aide de Composer. Une autre façon de découvrir vos dépendances est d'utiliser la commande "composer show".

## H - Introduction à Symfony Flex

Symfony Flex est un outil qui permet d'installer de nouvelles dépendances, il n'est pas obligatoire.

D'un point de vue technique, Symfony Flex est juste un plugin Composer. Flex est capable d'écouter les événements Composer, que ce soit l'installation, la mise à jour ou encore la suppression d'une dépendance.

Parmi les tâches qu'il est capable de réaliser :

- Appliquer une configuration par défaut pour un plugin Symfony.
- Créer des fichiers et des dossiers.
- Mettre à jour de fichiers (par exemple le fichier *config/bundles.php*).

Pour cela, Symfony Flex fonctionne à l'aide d'un système de "recettes" qui sont disponibles dans deux dépôts : un dépôt officiel maintenu par l'équipe Symfony et un dépôt communautaire ouvert à tous les mainteneurs de bundles, librairies et projets.

### SYMFONY UTILISE :

Le composant Dependency Injection et notamment comment construire des objets et les récupérer à l'aide du container de services :

**php bin/console debug :container = (ensemble des services de l'application)**

Le container de service est un objet qui est utilisé dans votre projet et auquel on a besoin d'accéder.

Ce service est enregistré dans un container, il est une "recette de cuisine", les étapes nécessaires à sa construction sont les suivantes : dépendances, méthodes et arguments à appeler.

Puisque les services sont présents dans le container de services, on peut les injecter sans crainte dans nos classes grâce à l'autowiring

## L'autowiring de services (auto câblage)

Cette fonctionnalité est activée par défaut dans tout projet Symfony 5.

Dans le fichier de configuration : config/services.yaml

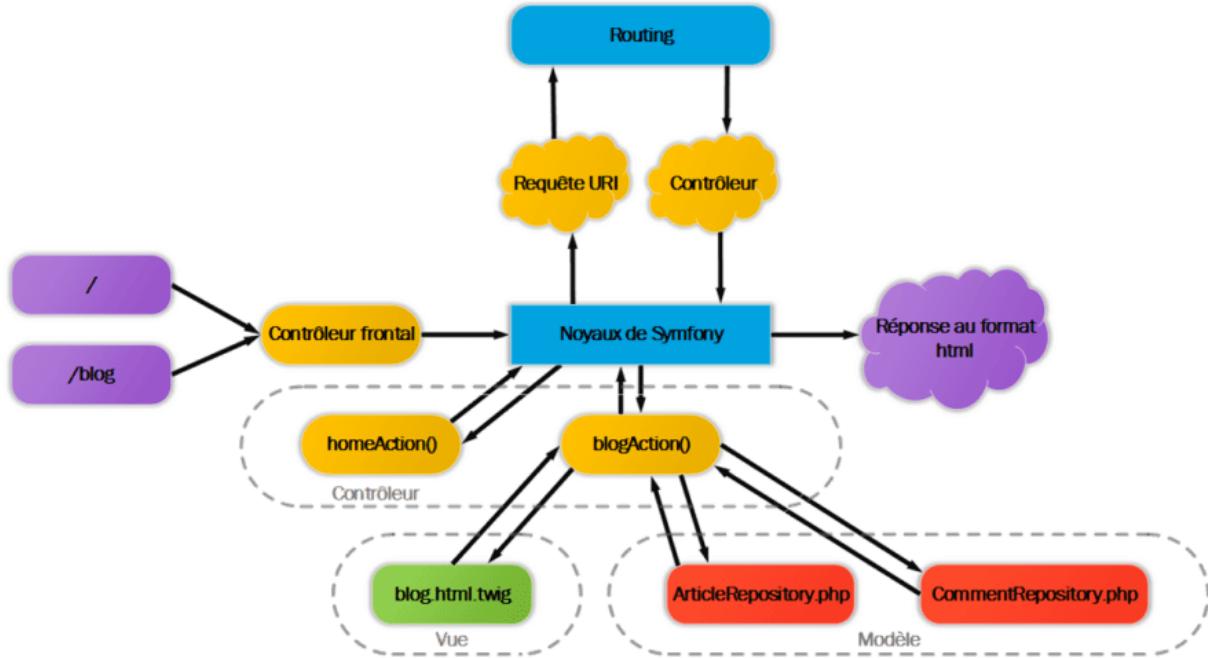
Doc Yaml : (langage normé de sérialisation équivalent à XML et Json)

[https://symfony.com/doc/5.4/components/yaml/yaml\\_format.html](https://symfony.com/doc/5.4/components/yaml/yaml_format.html)

Autowiring liste :

`php bin/console debug:autowiring`

(Ensemble des classes automatiquement charger par le Framework) = pas besoin de créer de service, c'est élément sont a injecté dans les paramètres des méthodes des contrôleurs.

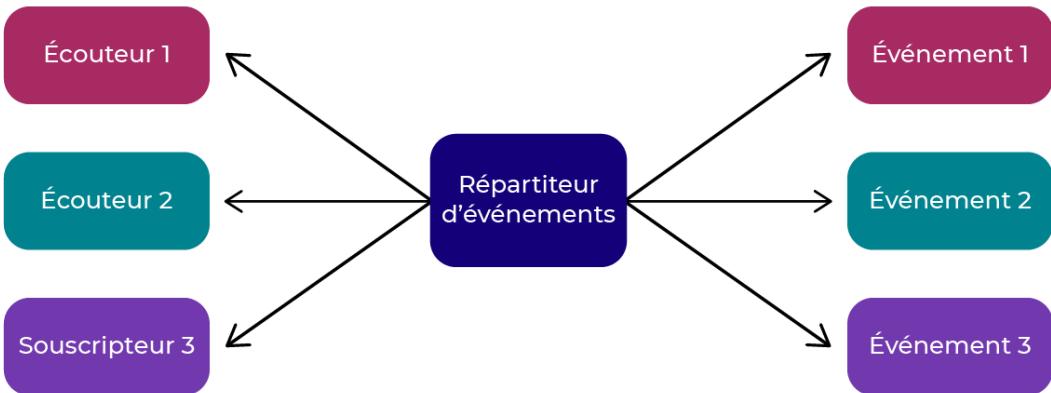


- Grâce à l'*autowiring*, l'essentiel du temps, nous n'avons rien de spécial à faire pour que nos objets soient automatiquement retrouvés par le container et accessibles dans nos services et nos contrôleurs.
- L'*autoconfiguration* permet d'ajouter des tags à nos services s'ils implémentent une interface spécifique et les services tagués sont traités différemment par le Framework.

Principe de Symfony :

### Le composant EventDispatcher en bref

Une application Symfony dispose d'un **répartiteur d'événements** qui va envoyer une série d'**événements** natifs et métiers. Ensuite, des objets, qui peuvent être des **écouteurs** ou encore des **souscripteurs d'événements**, peuvent écouter ces événements et exécuter des fonctions à partir de données qui sont transmises par l'événement.



- Écouteur 1 écoute l'événement 1, écouteur 2 l'événement 2, et le souscripteur l'événement 3.
- Les 3 "écouteurs" (2 écouteurs, 1 souscripteur) ont été ajoutés au répartiteur d'événements (ou encore "EventDispatcher").
- Quand le répartiteur envoie les événements, il donne l'information aux écouteurs qui peuvent donc réaliser des actions au bon moment sans pour autant avoir connaissance des autres écouteurs.

## I – Configurer une application

- Ajouter l'extension à PHPStorm **file -> setting -> plugins -> Symfony Supports**
- Le fichier. env (et test) regroupe toutes les variables d'environnement, elles sont utilisables grâce à la fonction helper\_env() et sont utilisable n'importe où dans le code (variable super globale Symfony)
- Lisent des variables d'environnement  
**php bin/console debug:container --env-vars**

## Symfony et MVC (Models Views Controllers)

- 1- Modèles = toutes les données logique métier BDD + Json + Calcul Mathématique + .txt, csv etc...
- 2- C'est le cœur de métier (utilisé avec ORM (Object relational mapping) DOCTRINE)
- 3- Views = Génère des pages HTML5 à renvoyée au client, partie visuelle regroupée dans le dossier /templates avec extension .html.twig

4- Controller = Bloc de code PHP qui coordonne l'application, ils appellent les modèles et les vues à retourner (Classe Request + Response)

## Introduction au make bundle : Symfony Maker

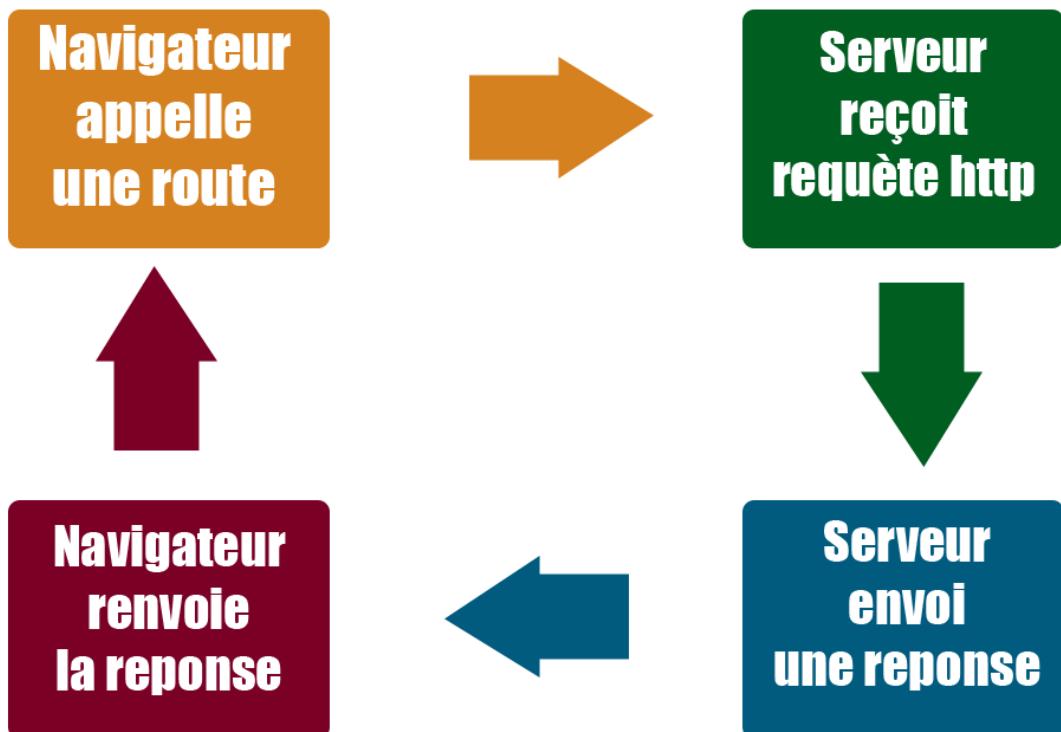
Symfony Maker vous aide à générer, des contrôleurs, des classes de formulaires, des tests et plus encore afin que vous puissiez oublier l'écriture de code partout.

Ce bundle est une alternative à [SensioGeneratorBundle](#) pour les applications Symfony modernes et nécessite l'utilisation de Symfony 3.4 ou plus récent.

Ce bundle suppose que vous utilisez une structure de répertoires Symfony 4 standard, mais de nombreuses commandes peuvent générer du code dans n'importe quelle application.

J – Les Contrôleurs : Dossier src / Controller

## LA LOGIQUE DU CONTROLLER



## LES CONTROLLERS

Dans le dossier : src/Controller

- 1- Un fichier PHP qui contient une classe
- 2- Ils doivent se terminer par Controller (ex : AccueilController)
- 3- On peut les créer à la main ou utiliser makerBundle

Maker-bundle est déjà présent dans l'installation de base de Symfony 5

Installer le bundle :

```
composer require symfony/maker-bundle --dev
```

Pour voir la liste des éléments disponibles depuis maker-bundle :

```
php bin/console list make
```

```
Available commands for the "make" namespace:  
make:admin:crud           Creates a new EasyAdmin CRUD controller class  
make:admin:dashboard       Creates a new EasyAdmin Dashboard class  
make:admin:migration       Migrates EasyAdmin2 YAML config into EasyAdmin 3 PHP config classes  
make:auth                  Creates a Guard authenticator of different flavors  
make:command               Creates a new console command class  
make:controller            Creates a new controller class  
make:crud                  Creates CRUD for Doctrine entity class  
make:docker:database        Adds a database container to your docker-compose.yaml file  
make:entity                Creates or updates a Doctrine entity class, and optionally an API Platform resource  
make:fixtures              Creates a new class to load Doctrine fixtures  
make:form                  Creates a new form class  
make:functional-test        Creates a new test class  
make:message               Creates a new message and handler  
make:messenger:middleware  Creates a new messenger middleware  
make:migration              Creates a new migration based on database changes  
make:registration-form     Creates a new registration form system  
make:reset-password         Create controller, entity, and repositories for use with symfonycasts/reset-password-bundle  
make:serializer:encoder     Creates a new serializer encoder class  
make:serializer:normalizer  Creates a new serializer normalizer class  
make:stimulus:controller    Creates a new Stimulus controller  
make:subscriber             Creates a new event subscriber class  
make:test                  [make:unit-test|make:functional-test] Creates a new test class  
make:twig-extension          Creates a new Twig extension class  
make:unit-test              Creates a new test class  
make:user                  Creates a new security user class  
make:validator              Creates a new validator and constraint class  
make:voter                 Creates a new security voter class
```

- a- Créer un Controller en ligne de commande :

```
php bin/console make :controller
```

Cette commande a pour effet de générer une classe Controller et un dossier + fichier index.html.twig dans le dossier templates

Exemple de :ProduitController.php

```
<?php

namespace App\Controller;

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

class ProduitsController extends AbstractController
{
    /**
     * @Route("/produits", name="produits")
     */
    public function index(): Response
    {
        return $this->render('produits/index', [
            'controller_name' => 'ProduitsController',
        ]);
    }
}
```

Explication :

- 1- Le namespace pour le polymorphisme : src/produits/ProduitsController
- 2- Import (via use = appel de classe depuis autoload.php => vendor) Classes AbstractController + Response + Route
- 3- La classe Produits Controller hérite d'Abstract Controller, qui donne accès à des méthodes communes à tous les Controller
- 4- La méthode index() attend en retour un élément de Type Response (return : Response Classe)
- 5- Les annotations `/** */` sont des paramètres de la classe Route
- 6- C'est un lien entre une requête envoyée par l'utilisateur et le nom de la méthode à exécuté dans un contrôleur.
- 7- **CE NE SONT PAS DES COMMENTAIRES**, ici lors de l'appel de la route /produits, le Framework exécutera la méthode index () et appellera la vue templates/produits/index.html.twig
- 8- Les annotations sont toujours écrites avant la méthode

- 9- Le paramètre name= 'nom de la route' de la route est la référence à URL /produits
- 10- La méthode index () exécute la méthode render () (issu d'Abstract Controller), elle permet de faire la jonction avec la vue index.html.Twig dans le dossier Template, elle transmet à la vue une variable controller\_name qui contient la valeur ProduitsController

Pour voir cette vue : **localhost :8000/produits**

Les composants HttpFoundation :

Tous accès à une application web se fait via une requête http, composée d'une en tête (header)

- Nom de domaine
- Type de contenus
- Statut
- Etc...

La requête est composée d'un Body dans lequel sont passé les paramètres à transmettre

La réponse du serveur à la même syntaxe sauf que le body renvoi de l'HTML interprétée par le navigateur

Dans Symfony les composants Request et Response sont contenus dans la bibliothèque HttpFoundation et ajouter grâce à l'autowiring

K-L 'Objet Request :

C'est un objet instancié à partir de la classe Request, cette classe est appelée directement dans les paramètres d'une méthode d'un contrôleur

Grâce à autowiring (il est donc directement instancié une seule fois)

Il faut donc appeler la classe en paramètre qui se charge d'instancier l'objet.

```

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

class ProduitsController extends AbstractController
{
    /**
     * @Route("/produits", name="produits")
     * @param Request $request
     * @return Response
     */
    public function index(Request $request): Response
    {
        echo $request->getPathInfo();
        return $this->render('produits/index', [
            'controller_name' => 'ProduitsController',
        ]);
    }
}

```

DE MANIERE GENERALE ON PEUT DIRE QUE :

Les classes (Objets Symfony) et leurs équivalents PHP :

- **Request** = `$_POST` en PHP (soumission des formulaires)
- **Query** = `$_GET` en PHP (paramètre transmit dans l'adresse de la requête)
- **Cookies** = `$_COOKIE` en PHP
- **Files** = `$_FILES` en PHP relative au fichier transmit
- **Server** = `$_SERVER` en PHP info serveur
- **Headers** retourne les entêtes des données des requêtes
- Toutes ces propriétés citées ci-dessus renvoient un objet de la `ParameterBag`
- On utilise ensuite les méthodes des classes pour récupérer des informations
- **All** = retourne tout
- **Keys** = retourne le nom des variables,
- **Get** ('nom de la variable') = retourne le nom d'une seule variable

- Has () retourne un booléen

Exemple : localhost :8000 /produits?info=premier requête&statut=message

```
/**  
 * @Route("/produits", name="produits")  
 * @param Request $request  
 * @return Response  
 */  
public function index(Request $request): Response  
{  
    echo $request->query->get('info');  
    return $this->render('produits/index', [  
        'controller_name' => 'ProduitsController',  
    ]);  
}
```

Pour afficher un tableau de valeur

```
print_r($request->query-all());
```

## L – L’Objet Response :

Il s’agit également d’un objet instancié à partir de la classe Response, cet objet permet de définir la réponse à envoyé au navigateur

Contrairement à Request on doit l’instancier à l’extérieur de l’action du contrôleur ou après la définition des paramètres de la méthodes ex :

Response est donc une valeur de retour d’une méthode d’un contrôleur

```
public function index(Request $request): Response
```

```

class ProduitsController extends AbstractController
{
    /**
     * @Route("/produits", name="produits")
     * @param Request $request
     * @return Response
     */
    public function index(Request $request)
    {
        $response = new Response( content: "Salut c Michael OLFP");
        return $response;
    }
}

```

Tout à l'heure la méthode \$this->render (page twig) génère implicitement l'objet Response

Chaque méthode d'un Controller retourne obligatoirement un objet Response

\$response permet d'appeler les méthodes json(), redirect(), generateUrl() et d'autres.

Exemple :

```

class ProduitsController extends AbstractController
{
    /**
     * @Route("/produits", name="produits")
     * @param Request $request
     * @return Response
     */
    public function index(Request $request)
    {
        $response = new Response();
        $response->setContent( content: "Salut c MIC");
        $response->headers->set( key: 'Content-Type', values: 'application/json');
        $response->setStatusCode( code: Response::HTTP_OK);
        $response->setCharset( charset: 'utf-8');
        return $response;
    }
}

```

Un exemple de Session et de redirection :

```

<?php

namespace App\Controller;

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

class ProduitsController extends AbstractController
{
    /**
     * @Route("/produits", name="produits", methods={"GET","POST"})
     * @param Request $request
     * @return Response
     */
    public function index(Request $request):Response
    {
        $session = $request->getSession(); // ceci = session_start()
        $session->set('nom_utilisateur','Michael Michel');
        return $this->redirectToRoute( route: 'redirection' );
    }

    /**
     * @Route("/produits/redirection", name="redirection")
     * @param Request $request
     * @return Response
     */
    public function redirection(Request $request){
        //Recup de la session
        $session = $request->getSession();
        $nomUtilisateur = $session->get( name: 'nom_utilisateur' );
        return new Response( content: "Ici la redirection + nom de session utilisateur $nomUtilisateur" );
    }
}

```

## M - Les FlashBags :

Ce sont des variables de session qui se supprime elle-même ce sont donc des éléments similaires aux notifications qui donnent des informations temporaires.

Ex : Dans le fichier templates/base.html.twig : on configure deux boucles pour afficher soit un message de succès, soit un message d'erreur.

```

<!-- LES FLASHBAGS = Notifications -->
{%
    for message in app.flashes('success') %}
        <div class="notification is-success">
            <button class="delete"></button>
            {{ message }}
        </div>
    {% endfor %}

{%
    for message in app.flashes('danger') %}
        <div class="notification is-danger">
            <button class="delete"></button>
            {{ message }}
        </div>
    {% endfor %}

```

Dans ProduitsController.php lors de l'ajout d'un produit et une condition if-else

On affiche une notification de succès ou d'erreur en fonction de l'état du programme

```

if(!is_string($file)){
    //On récupère le nom du fichier uploader
    $fileName = $file->getClientOriginalName();

    //déplacement de la photo = move_uploaded_file($_FILES['userfile']['tmp_name'] en php
    $file->move(
        //Destination du fichier configurer dans le dossier config/services.yaml => parameters
        //images_directory: '%kernel.project_dir%/public/img'
        //Ajouter la ligne a parameters : images_directory: '%kernel.project_dir%/public/img/'
        //En second paramètre = le nom du fichier
        $this->getParameter( name: "images_directory"),
        $fileName
    );
    //Attribution de la photo a l'entité a l'aide des setters
    $produit->setImageProduit($fileName);
    //Notification flash bag en cas de succès
    $this->addFlash( type: 'success', message: 'Votre annonce à bien été ajouté !');
} else{
    //Sinon Notification flash bag en cas d'erreur
    $this->addFlash( type: 'danger', message: 'Une erreur est survenue durant la création de votre annonce !');
    //redirection vers la page ajouter produits
    return $this->redirect($this->generateUrl( route: '/produits/new'));
}

```

Exemple de base dans un contrôleur avec une session et une redirection :

```

class ProduitsController extends AbstractController
{
    /**
     * @Route("/produits", name="produits", methods={"GET", "POST"})
     * @param Request $request
     * @return Response
     */
    public function index(Request $request):Response
    {
        $session = $request->getSession(); // ceci = session_start()
        $session->getFlashBag()->add( type: 'info', message: 'message info');
        $session->getFlashBag()->add( type: 'info', message: 'un autre message');
        $session->set('nom_utilisateur','Michael Michel');
        return $this->redirectToRoute( route: 'redirection');
    }

    /**
     * @Route("/produits/redirection", name="redirection")
     * @param Request $request
     * @return Response
     */
    public function redirection(Request $request){
        //Recup de la session
        $session = $request->getSession();
        $info = $session->getFlashBag()->get( type: 'info');
        $affiche = '';
        foreach ($info as $message) {
            $affiche .= $message.'<br />';
        }
        $nomUtilisateur = $session->get( name: 'nom_utilisateur');
        return new Response( content: "message : $affiche $nomUtilisateur");
    }
}

```

## N - LE SYSTEME DE ROUTE :

2 Types :

Méthode 1 : Des routes sans annotation

Un fichier de routes de situe dans : config/routes.yaml

```
#index:
#    path: /
#    controller: App\Controller\DefaultController::index
```

Les # sont des commentaires

Index : le nom de la méthode dans le contrôleur

Path : définis la route dans le navigateur

Controller : Le namespace App\Controller + le nom du fichier contrôleur + la méthode appelée

## Les routes dans votre fichier Controller

Dans un contrôleur : avant la déclaration de la méthode, on utilise des Annotations `@Route("/ma_route/ ")`

Cette syntaxe est particulière : elle ressemble à des commentaires :

```
/**  
 * @Route("/produits", name="app_produits_index", methods={"GET"})  
 * @param ProduitsRepository $produitsRepository  
 * @param PaginatorInterface $paginator  
 * @param Request $request  
 * @return Response  
 */
```

Les annotations peuvent prendre des paramètres ex :

name => au nom de la route

```
* @Route("/produits", name="produits", methods={"GET","POST"})
```

Rappel : **php bin/console debug:router**

Name	Method	Scheme	Host	Path
admin	ANY	ANY	ANY	/admin
app_produits_index	GET	ANY	ANY	/produits/
app_produits_new	GET POST	ANY	ANY	/produits/new
app_produits_show	GET	ANY	ANY	/produits/{id}
app_produits_edit	GET POST	ANY	ANY	/produits/{id}/edit
app_produits_delete	ANY	ANY	ANY	/produits/supprimer-produit/{id}
app_register	ANY	ANY	ANY	/register
app_login	ANY	ANY	ANY	/login
app_logout	ANY	ANY	ANY	/logout
_preview_error	ANY	ANY	ANY	/_error/{code}.{_format}
_wdt	ANY	ANY	ANY	/_wdt/{token}
_profiler_home	ANY	ANY	ANY	/_profiler/
_profiler_search	ANY	ANY	ANY	/_profiler/search
_profiler_search_bar	ANY	ANY	ANY	/_profiler/search_bar
_profiler_phpinfo	ANY	ANY	ANY	/_profiler/phpinfo
_profiler_search_results	ANY	ANY	ANY	/_profiler/{token}/search/results
_profiler_open_file	ANY	ANY	ANY	/_profiler/open
_profiler	ANY	ANY	ANY	/_profiler/{token}
_profiler_router	ANY	ANY	ANY	/_profiler/{token}/router
_profiler_exception	ANY	ANY	ANY	/_profiler/{token}/exception
_profiler_exception_css	ANY	ANY	ANY	/_profiler/{token}/exception.css

Liste toutes les routes de votre projet et leurs nom (Name) + méthode (Method) + le schéma (Scheme) + hôte (Host) + le chemin dans le navigateur (Path).

**GET pour les routes et POST pour les formulaires**

a- Passer des paramètres dans les routes elles même :

```
* @Route("/produits/{nom}/{prenom}", name="produits",
methods={"GET", "POST"})
```

Qui donne url : <http://localhost:8000/produits/mic/michel>

```
/**
 * @Route("/produits/{nom}/{prenom}", name="produits", methods={"GET", "POST"})
 * @param Request $request
 * @return Response
 */
public function index(Request $request, $nom, $prenom):Response
{
    $nom = "mic";
    $prenom = "michel";
```

Passer les valeurs des variables dans des paramètre de la méthode :

Pour url : <http://localhost:8000/produits>

```
/**
 * @Route("/produits/{nom}/{prenom}", name="produits", methods={"GET", "POST"})
 * @param Request $request
 * @return Response
 */
public function index(Request $request, $nom = 'LAGADEK', $prenom = 'BOB'):Response
{
    return new Response( content: "Bonjour $nom $prenom");
}
```

Retourne : Bonjour LAGADEK BOB

Passer des paramètres conditionnels = requirements

Ici un exemple avec des Regex (Regular Expression) Expression régulière

```
* @Route("/produits/{nom}/{prenom}", name="produits",
methods={"GET", "POST"}, requirements={"nom": "[a-z]{2-50}"})
```

Ici la variable \$nom doit être de type alphabétique et comprendre entre 2 et 50 caractères sinon la page retourne une erreur.

Les routes à appeler corresponde au paramètre name= 'ma\_route' (ici : produits), c'est ce paramètre qu'il faut appeler dans une ancre par exemple

Pour lister vos routes : php bin/console debug :router

## O- Les Vues : Dossier src / templates

1- La vue est un état final qui génère de HTML5 destinée à l'utilisateur

- 2- Elle possède l'extension .html.twig, Twig est un langage qui permet de faire des traitements dans la vue comme le ferai une page PHP, Twig permet d'utiliser des variables, faire des conditions, des boucles, des filtres. Ainsi les vues ne contiennent pas de PHP.
- 3- Lorsque l'on génère un Controller une vue est automatiquement générée
- 4- Dans le dossier templates, on peut y voir un fichier parent nommé base.html.twig qui est un layout de base commune à toutes les vues, c'est le squelette principal de votre application

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>% block title %}Le Mauvais Coin!{% endblock %}</title>
        <link rel="icon"
              href="data:image/svg+xml,<svg xmlns=%22http://www.w3.org/2000/svg%22
viewBox=%220 0 128 128%22><text y=%221.2em%22 font-size=%2296%22>•</text></svg>">
        {# Run Appel du cdn css du framework Bulma css #}
        {% block stylesheets %}
            <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css">
            <link rel="stylesheet" href="{{ asset('css/styles.css') }}>
        {% endblock %}

        {% block javascripts %}
            <script src="https://code.jquery.com/jquery-3.6.0.js"
                   integrity="sha256-H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk="
                   crossorigin="anonymous"></script>
        {% endblock %}
    </head>
    <body>
        {% block menu %}
            {% include "includes/menu" %}
        {% endblock %}

        {!-- LES FLASHBAGS = Notifications --}
        {% for message in app.flashes('success') %}
            <div class="notification is-success">
                <button class="delete"></button>
                {{ message }}
            </div>
        {% endfor %}

        {% for message in app.flashes('danger') %}
            <div class="notification is-danger">
                <button class="delete"></button>
                {{ message }}
            </div>
        {% endfor %}

        <script>
            document.addEventListener('DOMContentLoaded', () => {
                (document.querySelectorAll('.notification .delete') || []).forEach(($delete) => {
                    const $notification = $delete.parentNode;

                    $delete.addEventListener('click', () => {
                        $notification.parentNode.removeChild($notification);
                    });
                });
            });
        </script>

        {% block body %}
            {!-- ICI LE CONTENU DE CHAQUE VUE ENFANT DE CE FICHIER --}
        {% endblock %}
    </body>
</html>

```

Ce fichier reprend la structure de base d'une page HTML5 en y ajoutant des blocs de séparation

Dans la balise <head> on y ajoute des blocs Twig pour le css et le javascript

{% block stylesheet %} + {% endblock %}

Dans la balise <body> : on ajoute autant de bloc que l'on souhaite

## Exemple de la page index.html.twig

Dès la première ligne : on constate que le fichier index hérite de base.html.twig

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

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

{% block title %}Hello ProduitsController!{% endblock %}

{% block body %}
<style>
.example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
.example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
<h1>Hello {{ controller_name }}! </h1>

This friendly message is coming from:
<ul>
<li>Your controller at <code><a href="{{ 'C:/wamp64/www/symfony5/src/Controller/ProduitsController.php'|file_link(0) }}">src/Controller/ProduitsController.php</a></code></li>
<li>Your template at <code><a href="{{ 'C:/wamp64/www/symfony5/templates/produits/index.html.twig'|file_link(0) }}">templates/produits/index.html.twig</a></code></li>
</ul>
</div>
{% endblock %}
```

A - Ce fichier hérite du gabarit base.html.twig

B - Twig est composé de bloc avec la syntaxe {{ }} {% %} {# #}

C - Les blocs sont ouvrant et fermant {% block body %} {% endblock %}

D - Le code HTML doit être à l'intérieur d'un block

E – Toutes les vues héritent du template de base : base.html.twig avec :

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

P - TWIG un moteur de Template

<https://twig.symfony.com/doc/3.x/>

Pas de PHP dans vos vues.html.twig (twig est utilisable sans Symfony)

Chaque vue hérite du fichier template/base.html.twig :

On peut donc ajouter des appels CSS et JS dans les blocks {%stylesheets%} et JS comme ceci :

On appelle CDN du Framework Bulma CSS

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Le Mauvais Coin!{% endblock %}</title>
        <link rel="icon"
              href="data:image/svg+xml,<svg xmlns=%22http://www.w3.org/2000/svg%22
viewBox=%220 0 128 128%22><text y=%221.2em%22 font-size=%2296%22>●</text></svg>">
        {# Run Appel du cdn css du framework Bulma| css #}
        {% block stylesheets %}
            <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css">
        {% endblock %}

        {% block javascripts %}
            <script src="https://code.jquery.com/jquery-3.6.0.js"
                   integrity="sha256-H+K7U5CnXl1h5ywQfKtsj8PCmoN9aaq30gDh27Xc0jk="
                   crossorigin="anonymous"></script>
        {% endblock %}
    </head>
    <body>
        {% block body %}
            <!--ICI LE CONTENU DE CHAQUE VUE ENFANT DE CE FICHIER -->
        {% endblock %}
    </body>
</html>

```

Chaque vue hérite donc de Bulma css + jquery

La syntaxe :

**{ des valeurs }** = Afficher le contenu d'une variable ou le résultat d'une expression = Interpolation (issue de mustache, handleBar, Js, etc...)

**{% instruction %}** = Exécuter une structure de contrôle ou d'itération (if, for, etc...)

**{# des commentaires #}** = Commentaire Twig

Le debug dans une vue = var\_dump()

**{ dump(dernierProduit.nomProduit) }**

Exemple simple d'un contrôleur :

```

public function index(Request $request):Response
{
    $nom = 'Michael';
    $prenom = 'Michel';
    $age = 20;
    return $this->render('produits/index.html.twig',[

        'nom' => $nom,
        'prenom' => $prenom,
        'age' => $age
    ]);
}

```

```
]);  
}
```

Cette méthode retourne un appel à la vue via return  
render('chemin/fichier.html.twig') ;

Ici les paramètres de la méthode index () sont transmis via un tableau associatif  
(clé/valeur)

Pour afficher les variables dans index.html.twig entre {%- block body %} et {%- endblock %} on intercale les clés du contrôleur qui affiche la valeur

```
{% extends 'base.html.twig' %}  
  
{%- block title %}Hello ProduitsController!{%- endblock %}  
  
{%- block body %}  
    <h1>Bienvenue {{ nom }} {{ prenom }} tu as : {{ age }}</h1>  
{%- endblock %}
```

Exemple de condition : {%- if age > 15 %} {%- else %} {%- end if%}

```
{% extends 'base.html.twig' %}  
  
{%- block title %}Hello ProduitsController!{%- endblock %}  
  
{%- block body %}  
    <h1>Bienvenue {{ nom }} {{ prenom }} tu as : {{ age }}</h1>  
    {%- if age > 50 %}  
        <p>Tu as plus de 50 ans</p>  
    {%- else %}  
        <p>Tu as moins de 50 ans</p>  
    {%- endif %}  
{%- endblock %}
```

Il est possible comme en PHP d'inclure des fichiers

{%- include 'nom\_de\_la\_vue.html.twig' %} (par exemple dans base.html.twig on inclus menu.html.twig pour chaque page)

Ajouter une variable d'environnement :

Dans le fichier .env : on créer une variable d'environnement  
APP\_AUTHOR=MICHEL Michaël

Puis dans le fichier config/packages/twig.yaml : On ajoute notre variable

```
twig:  
    default_path: '%kernel.project_dir%/templates'  
    globals:  
        auteur: '%env(APP_AUTHOR)%'
```

Attention à l'indentation 4 espace et pas d'espace entre les %%

Enfin dans index.html.twig :

```
<h3>{{ auteur }}</h3> Affichera le contenu de la variable d'environnement
```

**Les variables de sessions avec twig:**

Elles sont appelées via app.

```
{{ app.session.get('nom_de_la_session') }}
```

Créer à la racine du dossier templates un fichier alert.html.twig :

```
<div class="container mt-5">  
    {% for message in app.session.flashBag.get('message') %}  
        <span class="alert alert-{{ app.session.get('statut') }}">  
            {{ message }}  
        </span>  
    {% endfor %}  
</div>
```

Puis dans la base base.html.twig (avant la fermeture du body):

```
{% include 'alert.html.twig' %}
```

Et le Controller pour tester :

```
public function index(Request $request):Response  
{  
    $session = $request->getSession();  
    $session->getFlashBag()->add('message', 'test des alerts');  
    $session->getFlashBag()->add('message', 'SECOND TEST ALERT');  
    $session->set('statut', 'success');  
    return $this->render('produits/index.html.twig');  
}
```

Du CSS et JS dans base.html.twig :

Dans le bloc {% block stylesheet %} et {% block javascript %}

Pour accéder à un élément du dossier public Twig utilise :

`{{ asset('dossier/fichier')}}` = public/css/fichier.css

<link rel='stylesheet' href='{{ asset(css/styles.css) }}' /> et

<script src='{{ asset('js/app.js') }}'></script>

C'est appel sont dupliquer sur toutes les pages qui hérite de base.html.twig

On utilise asset qui fait référence au dossier : votreProjet/public/...

Pour les liens le principe est le même :

Des liens <a href=""></a>

Puis dans twig <a href='{{ path('nom\_de\_la\_route') }}'>Liens</a>

Ici path fait référence a la valeur name de chaque annotation routes du contrôleur

Rappel pour lister les routes : `php bin/console debug :router`

Les filtres avec twig:

`{{ expression | filtre | filtre }}`

Transformer le texte en majuscules avec UPPER :

<h1>Salut a {{ nom | upper }}</h1>

Depuis votre contrôleur :

```
return $this->render('produits/index.html.twig',[  
    "Html5" => '<h3 class="text-danger">test de twig upper</h3>'  
]);
```

Puis dans Twig :

```
 {{ Html5 | raw }}
```

On récupère la clé du tableau associatif render + raw qui interprète les balises et classes HTML5

Des dates : variable + format de la date

```
{% Date_jour | date('d-m-Y à H:i:s') %}
```

Les fonctions Twig :

Pour une boucle for

```
{% for produit in pagination %}
```

Le code HTML

```
{% endfor %}
```

L'équivalent de var\_dump() = le debug dans twig

```
{% dump(Html5) %}
```

```
Résultat : "<h3 class="text-danger">TEST HTML RAW</h3>"
```

## R - La barre de debug Profiler

C'est une Web Tool Bar qui n'apparaît qu'en mode dev (définie dans .env)

```
composer require --dev symfony/profiler-pack
```



Statut + Route + Temps de chargement + Mémoire + Cache + Profile + Twig

Beaucoup d'informations sont contenues dans le debugger, notamment : le contenu des objets Request et Response, les formulaires, les erreurs, les logs, les événements, les routes, le cache, les traductions, la sécurité, etc...

## S - Symfony Flex dans le détail

Installer par défaut lors de l'installation --full :

```
symfony new --full my_project
```

Il permet d'installer des dépendances comme composer (composer.json), il utilise des recette (recipes) qui sont des dépôts du site : <https://packagist.org/>

Toutes les dépendances Symfony se situent dans le fichier composer.json et tous les bundles du noyau Symfony (Kernel) sont dans le fichier config/bundles.php

Pour résoudre des problèmes de norme de codage :

**composer require cs-fixer**

## T -La couche modèle avec Doctrine (ORM)

Object Relational Mapping

**Les bases de données** = fichier structurée de données destiné à être extrait pour avoir accès à une information de manière sécurisée

Les données sont structurées sous forme de table appelée entité.

Symfony propose de base :

- SQLite
- MySQL
- PostgreSQL

**SQL** = (Structured Query Language ):

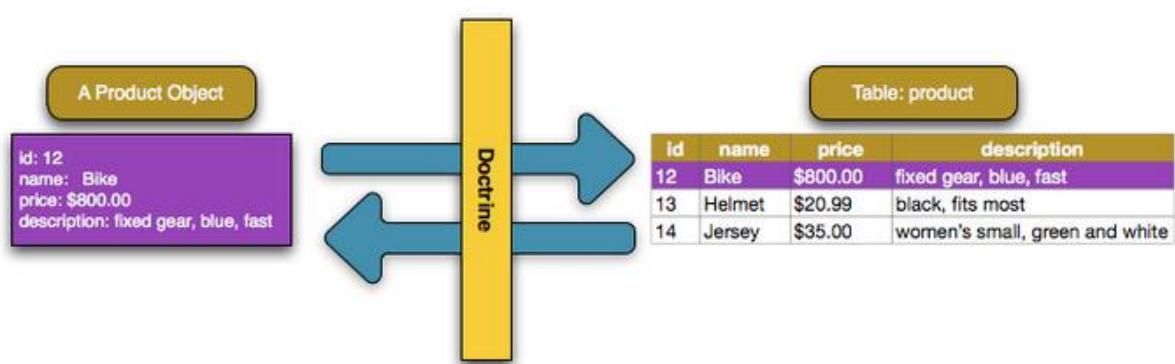
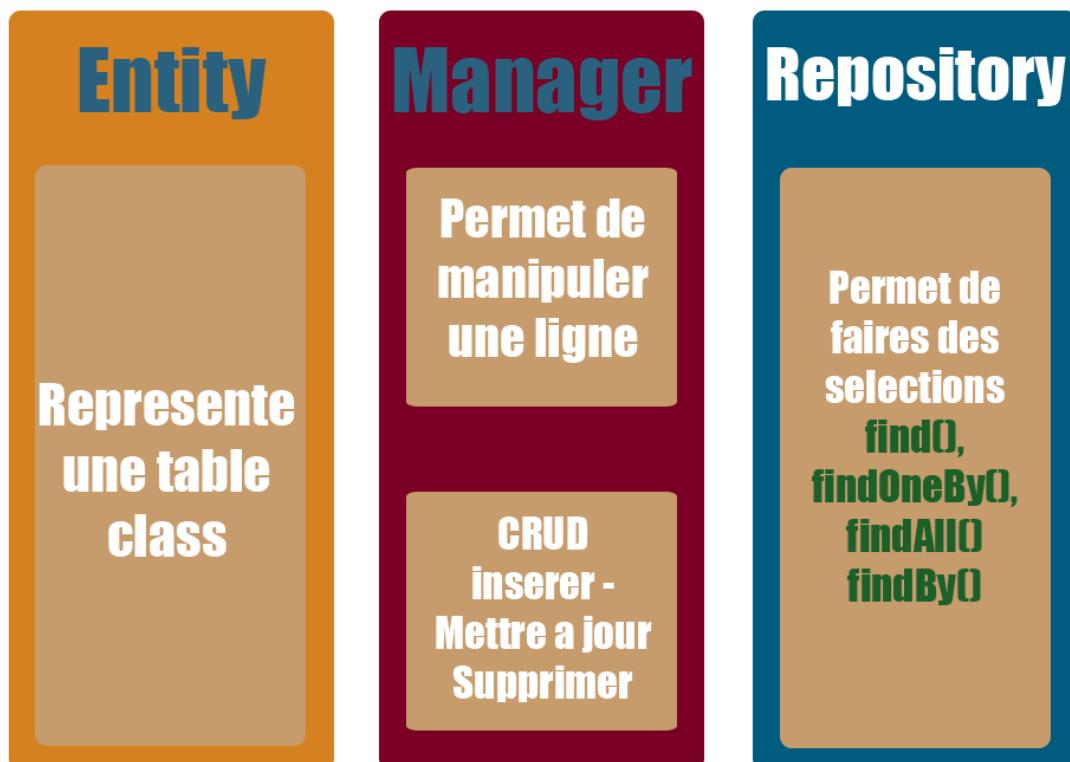
**CRUD** = Create Read Update Delete

Symfony dispose de Doctrine (**ORM** = Object Relational Mapping)

Il faut donc simplement utiliser des classes de Doctrine au lieu d'écrire du SQL

## U - DOCTRINE ORM ET DQL :

### DOCTRINE ORM SYMFONY Object Relational Mapping



## V – Le fichier de configuration .env et MySQL :

Le fichier .env contient des variables globales d'environnement utilisable partout.

Dans le fichier .env décommenter la ligne suivante :

Ce paramètre implique de disposer de WampServer (Combo Windows + PHP + Apache + MySQL)

Ligne par défaut dans .env :

```
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7&charset=utf8mb4"
```

Vous avez donc quelque chose de similaire à une connexion PDO :

```
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
```

Pour Symfony :

mysql:host utilisateur PhpMyAdmin + Mot de passe PhpMyAdmin + IP Locale + port MySQL + Nom de base de données + version du serveur

```
DATABASE_URL="mysql://root@127.0.0.1:3306/symfony5?serverVersion=5.7&charset=utf8mb4"
```

W – Créer une base de données :

```
php bin/console doctrine :database :create
```

Par défaut la base de données créée, prend le nom du paramètre décrit dans le fichier .env (ici : symfony5)

Vérifié la création de la base de données à l'adresse :

```
http://localhost/phpmyadmin
```

X – Créer une entité (Table et / ou Classe) et des migrations (fichier script) :

Créer une entité

```
php bin/console make :entity
```

Répondre aux questions :

Nom de l'entité : Produits

Les champs : (Pas besoin créer de clé primaire Symfony le fait pour vous)

- Nom\_produit string 255
- Description\_produit text
- Image\_produit string 255
- Stock\_produit boolean
- Date\_depot\_produit datetime

Réaliser une migration :

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

Une migration fait passer votre base de données d'un état A => B :

Les migrations exécutent des requêtes DQL (Doctrine Query Langage)

Puis : le Flush (exécution du DQL du dernier fichier de migration) :

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

Ceci a pour effet de créer un dossier src/Entity/Produits.php et Repository/ProduitRepository et un dossier + fichier de migration avec une version

## Y – Entity/Produit.php

Ce fichier définit l'entité produit toutes ses propriétés et ses accesseur et Mutateur (Getter & Setter) similaire au langage Java

Ligne de commandes

**php bin/console make:entity**

Entrer le nom de votre entité (C'est une classe donc une majuscule)

ex : **Produits**

Ajouter votre premier champ :

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

ex : **nom\_produit**

A CE STADE ENTRER : ?

Cette étape permet de connaître tous les types de champ proposé par Symfony

**PAS BESOIN DE CREER DE CHAMP CLE PRIMAIRE SYMFONY LE FAIT POUR VOUS**

Ce champ est de type :

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

Longueur du champ :

Field length [255]:

Le champ peut être null :

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

On vous demande si vous voulez ajouter un autre champ :

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

Si vous avez terminé : valider avec la touche entrée du clavier

**EN CAS D'ERREUR : PAS DE PANIQUE VOTRE ENTITE PEUT ETRE MODIFIER À TOUT MOMENT**

## Les migrations

Pour créer des entité (table) Symfony passe par un stade de migration (classe qui décrit comment faire l'opération)

Pour exécuter une migration

**php bin/console make:migration**

PUIS :

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

Ceci a pour effet de générer un fichier de migration dans le dossier migrations

Ex : Dossier + fichier : migrations/Version21458251.php

```
public function up(Schema $schema) : void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('CREATE TABLE produit (id INT AUTO_INCREMENT NOT NULL,
nom_produit VARCHAR(255) NOT NULL, prix_produit DOUBLE PRECISION NOT
NULL, quantite_produit INT NOT NULL, rupture TINYINT(1) NOT NULL, PRIMARY
KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci`'
ENGINE = InnoDB');
}

public function down(Schema $schema) : void
{
    // this down() migration is auto-generated, please modify it to your needs
}
```

```
    $this->addSql('DROP TABLE produit');
}
```

Ces opérations sont des requêtes SQL (DQL) classique, les méthodes down et up permet d'avancer ou de retourner en arrière

Vérifié la création de votre table dans PhpMyAdmin :  
<http://localhost/phpmyadmin/>

Pour revenir en arrière (migration précédente) :

**php bin/console doctrine :migration :migrate prev**

#### 4 Alias pour les migrations

- first = Migrez avant la 1<sup>ère</sup> version
- prev = Migrez avant la version précédente
- next = Migrez vers la prochaine version
- latest = Migrez à la dernière version

#### LES FICHIERS DANS LE DOSSIER src/entity/Produits.php

Chaque entité (table dans PhpMyAdmin) générée par la ligne de commande créer une classe avec des propriétés et des méthodes

Les propriétés sont les champs de la table + types

Les méthodes sont des getters (accesseur) et des setters (mutateur)

Chaque propriété possède donc sa méthode pour accéder et modifier un champ de la table

Chaque entité est liée à un Repository, c'est une classe + méthode capable d'effectuer des requêtes SQL pour accéder et modifier une entité.

#### src/Entity/Produits.php

Ce fichier dispose également d'annotation pour chaque propriété, elles précisent les informations utiles de la table produits au sein de la base de données Symfony5.

```
/**  
 * @ORM\Entity(repositoryClass=ProduitsRepository::class)  
 */
```

@ORM : Décrit le chemin vers le fichier ProduitsRepository qui fera les requêtes SQL (DQL = Doctrine Query Language)

```
/**  
 * @ORM\Id  
 * @ORM\GeneratedValue  
 * @ORM\Column(type="integer")  
 */  
private $id;
```

@ORM\GeneratedValue () définit id comme clé primaire, auto incrémentée de type entier (Integer)

```
/**  
 * @ORM\Column(type="string", length=255)  
 */  
private $nomProduit;
```

Le champ nom du produit de type string = chaîne de caractère 255 octets

Etc...

Puis chaque champ possède ses accesseurs et mutateur (Getter et Setter)

## Z – Fixtures ou faux jeux de données

Remplir une table avec un jeu de fausses données est simple avec Symfony, elles sont réalisables dans le Controller ou en dehors grâce à la commande :

Installation du bundle: **composer require --dev orm-fixtures**

Le drapeau (flag) –dev stipule que les fixtures sont ajoutées seulement pour le mode développement (.env = dev)

Toutes les fixtures héritent de la classe Fixtures de Fixture Bundle (classe abstraite) et sont chargée grâce à la méthode abstraite load()

La ligne de commande a pour effet de créer le dossier src/DataFixtures + un fichier ProduitFixture.php

## METHODE 1 : FAKER

Créer une entité Articles : **php bin/console make :entity**

Articles

Les champs :

nom\_article 255 string

contenu\_article text

image\_article 255 string

auteur\_article 255 string

date\_depot\_article dateTime

Validation : **php bin/console make:migration**

Puis : **php bin/console doctrine :migrations :migrate**

Generer un crud : **php bin/console make :crud**

Insatller Faker : <https://github.com/fzaninotto/Faker>

**composer require fzaninotto/faker**

Puis : **php bin/console make:fixtures**

ArticlesFixtures

Le code dans DataFixtures/ArticlesFixtures.php

```
<?php

namespace App\DataFixtures;

use App\Entity\Articles;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;
use Faker;
class ArticlesFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        //Appel de faker
        $faker = Faker\Factory::create('fr_FR');
```

```
//Creation d'un tableau vide
$articles = Array();
//Boucle soit 20 elements
for ($i = 0; $i < 20; $i++) {
    //Instancie de entité class
    $articles[$i] = new Articles();
    //Jeu de fausse donnée
    $articles[$i]->setNomArticle($faker->word);
    $articles[$i]->setContenuArticle($faker->sentence($nbWords = 6,
$variableNbWords = true));
    $articles[$i]->setImageArticle($faker->imageUrl($width = 640,
$height = 480));
    $articles[$i]->setAuteurArticle($faker->lastName);
    $articles[$i]->setDateArticle($faker->dateTime($max = 'now',
$timezone = null));
    $manager->persist($articles[$i]);
}
$manager->flush();
}
```

Puis : **php bin/console doctrine:fixtures:load**

Pour focus sur un fichier particulier :

**php bin/console doctrine:fixtures:load --group=ArticlesFixtures --append**

Votre entité articles est remplie

Effectuer la même opération pour votre entité (Table) Produits

```
4
5     use App\Entity\Produits;
6     use Doctrine\Bundle\FixturesBundle\Fixture;
7     use Doctrine\Persistence\ObjectManager;
8     use Faker;
9
10    class ProduitsFixtures extends Fixture
11    {
12        public function load(ObjectManager $manager): void
13        {
14            //Instance de la classe faker
15            $faker = Faker\Factory::create(locale: 'fr_FR');
16            //Un variable stock un tableau de valeur
17            $produits = array();
18            //Boucle pour 20 faux articles
19            for ($i = 0; $i < 20; $i++) {
20                //Instance de la classe Articles
21                $produits[$i] = new Produits();
22                //Jeu de fausse données champs par champs
23                //Le tableau-> le mutateur (setter de Article entity) + (paramètres faker)
24                $produits[$i]->setNomProduit($faker->word);
25                $produits[$i]->setDescriptionProduit($faker->sentence($nbWords = 6, $variableNbWords = true));
26                $produits[$i]->setImageProduit($faker->imageUrl($width = 640, $height = 480));
27                $produits[$i]->setStockProduit(stock_produit: "true");
28                $produits[$i]->setDateDepotProduit($faker->dateTime($max = 'now', $timezone = null));
29                $produits[$i]->setPrixProduit($faker->randomFloat($nbMaxDecimals = NULL, $min = 0, $max = NULL));
30                //Entity manager de Doctrine va faire persister les fausse donnée
31                $manager->persist($produits[$i]);
32            }
33        }
34        $manager->flush();
35    }
36}
37
```

En cas d'erreur d'entité :

Modifié les champs de votre entité et mettre à jour le schéma

**php bin/console doctrine :schema:update –force**

Puis :

**php bin/console doctrine:cache:clear**

Z-1 – Créer un contrôleur et afficher les données de la table produit

**A CE STADE VOUS POUVEZ DEJA GENERER UN CRUD  
AUTOMATIQUEMENT**

**php bin/console make :crud**

**POUR DES RAISON D'APPRENTISSAGE CES ETAPES VONT ETRE  
REALISÉ A LA MAIN**

Dans un contrôleur la récupération de données ce fait également avec Doctrine

```
php bin/console make :controller ProduitsController
```

Cette ligne de commande a pour effet de créer un dossier `src/controller` fichier `ProduitsController` et un dossier `produit` + une vue dans `templates/produits/index.html.twig`

Pour récupérer les données on utilise Doctrine EntityManager et le `ProduitRepository` et la méthode `findAll()`

LE REPOSITORY Produits :

Lors de la génération de l'entité (migration) : Symfony a généré un fichier `ProduitsRepository` pour exécuter des requêtes SQL (DQL)

Par défaut 4 méthodes de base sont accessibles :

```
<?php

namespace App\Repository;

use App\Entity\Produits;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;
use function Doctrine\ORM\QueryBuilder;

/**
 * @extends ServiceEntityRepository<Produits>
 *
 * @method Produits|null find($id, $lockMode = null, $lockVersion = null)
 * @method Produits|null findOneBy(array $criteria, array $orderBy = null)
 * @method Produits[] findAll()
 * @method Produits[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class ProduitsRepository extends ServiceEntityRepository
```

`find()` : retourne l'entité en fonction de son id

`findAll()` : retourne toutes les valeurs de l'entité

`findBy()` : retourne l'entité en fonction de la valeur d'un paramètre

`findOneByd()` : retourne un seul résultat de l'entité

Afficher les produits avec : `ProduitsController.php`

```

/**
 * @Route("/", name="app_produits_index", methods={"GET"})
 * @param ProduitsRepository $produitsRepository
 * @param PaginatorInterface $paginator
 * @param Request $request
 * @return Response
 */
public function index(ProduitsRepository $produitsRepository, PaginatorInterface $paginator, Request $request): Response
{
    //Appel du service PaginatorInterface en paramètre
    //Appel de la méthode paginate + paramètres

    $pagination = $paginator->paginate(
        //On récupère tous les articles

        $produitsRepository->findAll(),

        //On liste par entier (knp_paginator.yaml) on définit la clé dans url, par défaut ma page=1 + nombre d'article à afficher (ici 2)
        $request->query->getInt('page', default: 1), limit: 3
    );

    return $this->render('produits/index', [
        'controller_name' => 'ProduitsController',
        'produits' => $produitsRepository->findAll(),
        'dernierProduit' => $produitsRepository->getDernierProduit(),
        'pagination' => $pagination,
    ]);
}

```

On passe donc la classe ProduitsRepository en paramètre de la méthode et après le render (appel de la vue), dans un tableau associatif :

On créer une clé produits qui est égale à la classe ProduitsRepository-> + la méthode findAll() ;

### LA VUES : templates/produits/index.html.twig

Twig récupère la clé ‘produits’ du contrôleur et grâce à l’interpolation, on récupère les Getters de l’entité qui sont interpolé par le langage twig

`{{produit.nomProduit}}`

On créer donc une boucle for avec un alias de ‘produits’ du contrôleur

`{% for produit in produits %}`

Mise en page Bulma card + appel des données avec Twig :

**(AJOUTER LE CDN BULMA AU templates/base.html.twig)**

La vue twig qui affiche les produits (templates/produits/index.html.twig)

```

  1  {# Hérite du template parent #}
  2  {% extends 'base.html.twig' %}

  3  {%
  4  | block title %}Le Mauvais Coin{%
  5  | endblock %}

  6  {%
  7  | block body %}
  8  <div class="container">
  9    <div class="columns is-multiline">
 10      {% for produit in produits %}
 11        <div class="column is-4">
 12          <div class="card">
 13            <div class="card-image">
 14              <figure class="image is-4by3">
 15                
 16              </figure>
 17            </div>
 18            <div class="card-content">
 19              <div class="media">
 20                <div class="media-left">
 21                  <figure class="image is-48x48">
 22                    
 23                  </figure>
 24                </div>
 25                <div class="media-content">
 26                  <p class="title is-4">{{ produit.nomProduit }}</p>
 27                  <p class="subtitle is-6">PRIX : {{ produit.prixProduit }} € </p>
 28                </div>
 29              </div>
 30            <div class="content">
 31              {{ produit.descriptionProduit }}
 32              <a href="#">#css</a> <a href="#">#responsive</a>
 33              <br>
 34              <p>Date de dépôt : {{ produit.dateDepotProduit | date('d/m/Y') }}</p>
 35              <div class="card-action">
 36                <a href="#">Détails du produit</a>
 37              </div>
 38            </div>
 39          </div>
 40        </div>
 41      {% endfor %}
 42    </div>
 43  {%
 44  | endblock %}

```

On utilise donc une instruction Twig avec une boucle for et le paramètres ‘liste Produit’ du tableau associatif du contrôleur :

{% for produit in produits %}

{% endfor %}

Pour afficher chaque élément on appelle l’attribut de notre boucle produit.Getter soit :

`{{produit.nomProduit}}` = le getter de l’entité Produits soit : `getNomProduit()`

```

public function getNomProduit(): ?string
{
    return $this->nom_produit;
}

```

Résultat :



**Chariot à roulette**  
Prix : 573.02 €

Référence du produit (clé étrangère numéro) : 74586  
Catégorie : (clé étrangère ManyToOne & OneToMany) : Matériel Pro  
Distributeur(s) : eBay  
Nom du vendeur : michael@gmail.com

[Plus d'info](#)



**Iphone**  
Prix : 458.25 €

Référence du produit (clé étrangère numéro) : 12586  
Catégorie : (clé étrangère ManyToOne & OneToMany) : Emploi  
Distributeur(s) : Amazon  
Distributeur(s) : LeBonCoin  
Nom du vendeur : michael@gmail.com

[Plus d'info](#)



**Lego**  
Prix : 50.25 €

Référence du produit (clé étrangère numéro) : 784589  
Catégorie : (clé étrangère ManyToOne & OneToMany) : Loisirs  
Distributeur(s) : LeBonCoin  
Distributeur(s) : eBay  
Nom du vendeur : michael@gmail.com

[Plus d'info](#)

## Z-2 - AFFICHER UNE PAGINATION :

Utilisé le bundle : <https://github.com/KnpLabs/KnpPaginatorBundle>

Installation : composer require knplabs/knp-paginator-bundle

Créer un fichier config/packages/knp\_paginator.yaml

```
knp_paginator:  
    page_range: 5 # Nombre de liens montre de la  
pagination  
    default_options:  
        page_name: page # Nom de la clé query paramètre  
        sort_field_name: sort # paramètre query du champs de tri  
        sort_direction_name: direction # direction du tri  
        distinct: true # Utile en cas de group_by  
        filter_field_name: filterField # champ de filtre  
        filter_value_name: filterValue # valeur du filtre  
    template:  
        pagination: '@KnpPaginator/Pagination/bulma_pagination.html.twig'  
        # ici le template bulma = vendor/knp-paginator-bundle-src-template/  
        sortable: '@KnpPaginator/Pagination/sortable_link.html.twig' # sort  
link template  
        filtration: '@KnpPaginator/Pagination/filtration.html.twig' # filters  
template
```

Dans le contrôleur ProduitsController.php, on modifie la méthode de liste de produits

```
/**  
 * @Route("/", name="app_produits_index", methods={"GET"})  
 * @param ProduitsRepository $produitsRepository  
 * @param PaginatorInterface $paginator  
 * @param Request $request  
 * @return Response  
 */  
public function index(ProduitsRepository $produitsRepository, PaginatorInterface $paginator, Request
```

```

$request): Response
{
    //Appel du service PaginatorInterface en paramètre
    //Appel de la méthode paginate + paramètres
    $pagination = $paginator->paginate(
        //On récupère tous les articles
        $produitsRepository->findAll(),
        //On liste par entier (knpPaginator.yaml) on définit la clé dans url, par défaut ma page=1 + nombre
        //d'article à afficher (ici 2)
        $request->query->getInt('page', 1), 2
    );

    return $this->render('articles/index.html.twig', [
        'controller_name' => 'ArticlesController',
        'articles' => $produitsRepository->findAll(),
        'pagination' => $pagination
    ]);
}

```

## Traduire les boutons en Fr

Les fichiers de traductions sont déjà fournis par le bundle de pagination :

Vendor/knplabs/knp-paginator-bundle/translation/fr .xliff

Il faut simplement passer le Framework en fr :

Config/packages/translation.yaml :

```

framework:
    default_locale: fr
    translator:
        default_path: '%kernel.project_dir%/translations'
        fallbacks:
            - fr

```

## Effacer le cache :

La commande : php bin/console doctrine :cache:clear

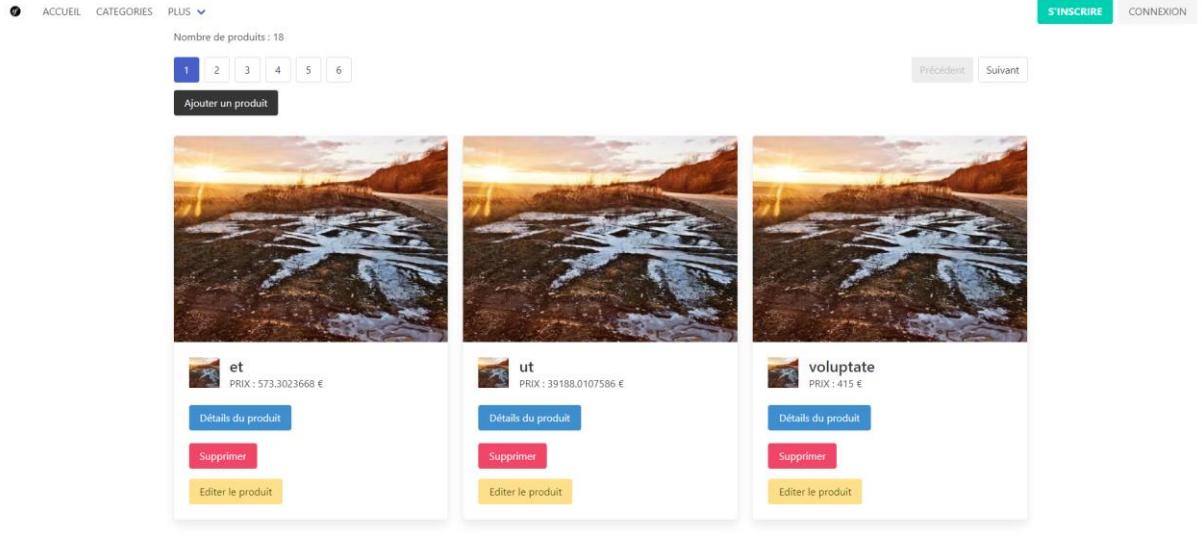
## La vue :

```

<!--LA PAGINATION-->
<div class="count">
    Nombre de produits : {{ pagination.getTotalItemCount }}
</div>
<div class="navigation" style="margin-top: 20px">
    {{ knp_pagination_render(pagination) }}
</div>

```

Résultat :



## Z-3 – LES DETAILS D’UN PRODUIT

Pour afficher les détails de l’article, on ajoute une méthode et une route avec un paramConverter capable de récupérer les informations de notre entité

On passe dans url un {id}

```
@Route("/produits/{id}")
```

Puis le code de la méthode detailsProduits dans ProduitsController

```
/*
 * @Route("/produits/{id}", name="app_details_produits", methods={"GET"})
 */
//En paramètres l’entité Produits
public function detailsProduits(Produits $produits):Response{
    return $this->render('liste_produits/detailsProduit.html.twig',[

        'produit' => $produits
    ]);
}
```

Focus sur la route :

**La vue :**

Ici la clé ‘produit’ sera utilisée dans la vue twig pour afficher les détails de chaque produit dans url : exemple <http://localhost:8000/produits/21>

```
①  {% extends 'base.html.twig' %}

②  {% block title %}Détail articles{% endblock %}

③  {% block body %}
7   <div class="columns is-mobile is-centered" style="margin-top: 20px; padding: 20px">
    <div class="column is-narrow">
        <div class="card">
            <div class="card-image center">
                
            </div>
            <div class="card-content">
                <h3 class="title is-3">{{ produit.nomProduit }}</h3>
                <p>{{ produit.descriptionProduit }}</p>
                <em class="orange-text">Posté le : {{ produit.dateDepotProduit | date('d/m/Y') }}</em>
                <p>PRIX: {{ produit.prixProduit }} €</p>
                <p class="card-text">Produit en stock : <em>{{ produit.stockProduit ? 'OUI' : 'NON' }}</em></p>
            </div>
            <div class="card-action">
                <a class="button is-warning m-3" href="{{ path('app.liste_produits') }}>Retour</a>
            </div>
        </div>
    </div>
</div>

    &lt;% endblock %&gt;
```



## ut

Quia aut molestiae tenetur mollitia commodi ut.

Posté le : 30/07/1992

PRIX: 2134144 €

Produit en stock : OUI

[Retour](#)

## Z-4 – SUPPRIMER UN PRODUIT

Pour supprimer un produit :

Le contrôleur a besoin de 4 paramètres

- La classe Request
- L'entité Produits
- Le repository Produits (requêtes DQL)
- La classe Réponse

Le contrôleur va vérifier la validité d'un jeton unique à chaque produit : si ce jeton est valide => on supprime le Produits

Etapes en 2 temps :

Le contrôleur check le jeton de la vue \_delete.form.html.twig

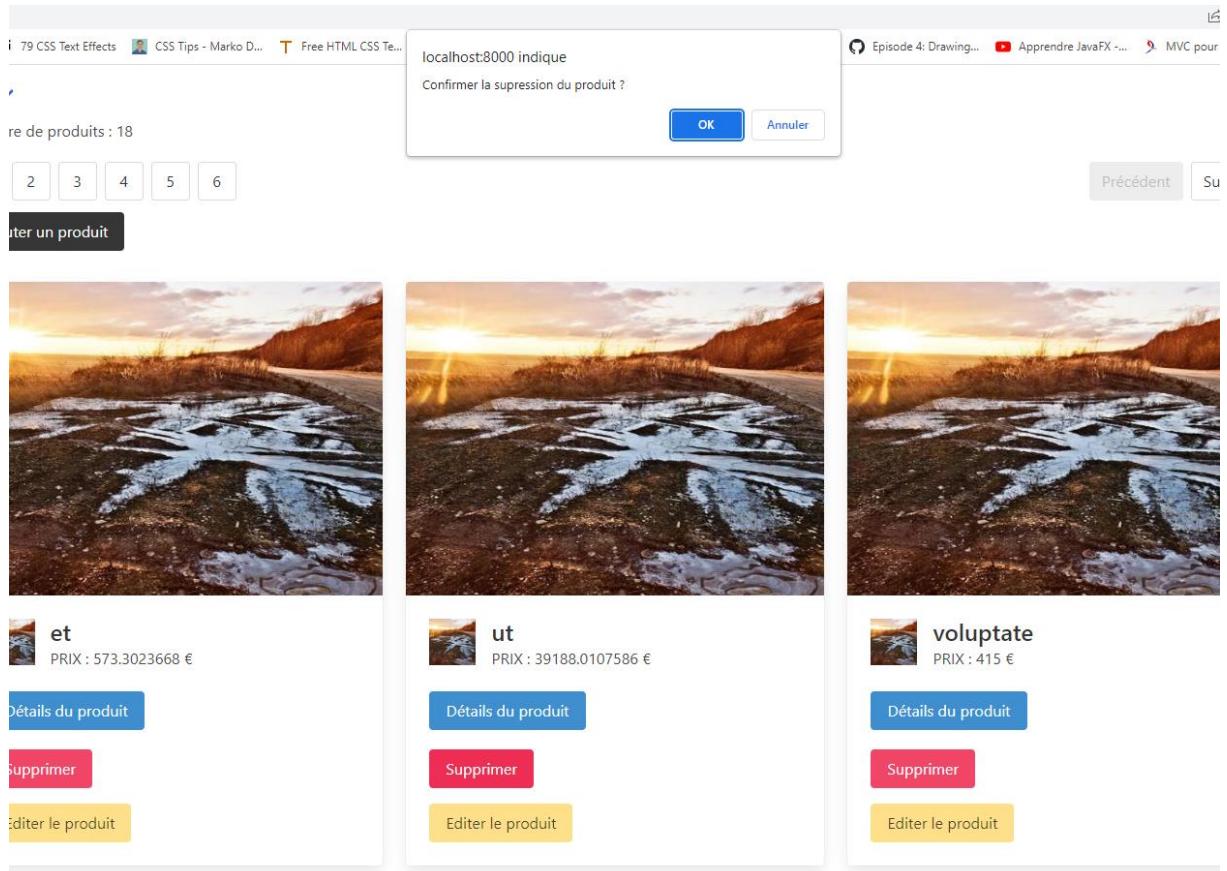
Le contrôleur : On utilise la méthode remove() du repository

```
/*
 * @Route("/supprimer-produit/{id}", name="app_produits_delete")
 * @param Request $request
 * @param Produits $produit
 * @param ProduitsRepository $produitsRepository
 * @return Response
 */
//On utilise la classe Request + entité produit + le repository = requête DQL + classe Response
public function delete(Request $request, Produits $produit, ProduitsRepository $produitsRepository): Response
{
    //check le _token du fichier _delete_form.html.twig
    if ($this->isCsrfTokenValid('delete', $produit->getId(), $request->request->get('_token'))) {
        //Si ca match on supprime le produit de l'entité
        $produitsRepository->remove($produit, true);
    }
    //on effectue une redirection vers la page produit
    return $this->redirectToRoute('produits', [], Response::HTTP_SEE_OTHER);
}
```

La vue : \_delete.form.html.twig

```
<form method="post" action="{{ path('produits/supprimer-produit', {'id': produit.id}) }}" onsubmit="return confirm('Confirmer la suppression du produit ?');">
    <input type="hidden" name="_token" value="{{ csrf_token('delete') ~ produit.id }}>
    <button class="button is-danger">Supprimer</button>
</form>
```

A la suppression une alerte popup est déclenché et demande à l'utilisateur de valider la suppression



## Z-5 – AJOUTER UN PRODUIT

Pour appliquer un thème bulma au formulaire :

- Récupéré le fichier twig suivant :
- [https://raw.githubusercontent.com/dsmink/twig-bulma-form-theme-bundle/master/views/Form/bulma\\_0\\_3\\_x\\_layout.html.twig](https://raw.githubusercontent.com/dsmink/twig-bulma-form-theme-bundle/master/views/Form/bulma_0_3_x_layout.html.twig)
- Placer le contenu dans un fichier templates/bulma\_form.html.twig
- Ouvrir le fichier : config/packages/twig.yaml
- Ajouter la ligne :

```
twig:
  default_path: '%kernel.project_dir%/templates'
  form_themes: ['bulma_form.html.twig']
```

VOS FORMULAIRE ONT TOUS LES CLASSES CSS DU FRAMEWORK BULMA

## LES FORMULAIRES

La création et le traitement de formulaires HTML sont difficiles et répétitifs. Vous devez vous occuper du rendu des champs de formulaire HTML, de la validation des données soumises, du mappage des données du formulaire dans des objets et bien plus encore.

Symfony inclut une fonctionnalité de formulaire puissante qui fournit toutes ces fonctionnalités et bien d'autres pour des scénarios vraiment complexes.

<https://symfony.com/doc/5.4/forms.html>

Chaque fichier doit être nommé avec le mot clé Type

Ex : ProduitsType.php

Généré des formulaires :

**php bin/console make :form**

Le crud générer par Symfony à créer un dossier src/Form/ProduitsType.php

Ces fichiers utilisent la classe formBuilder qui spécifie chaque type de champs

Exemple de ProduitsType.php

Ajouter un système d'upload de photo au contrôleur :

```
class ProduitsType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('nom_produit', TextType::class)
            ->add('description_produit', TextareaType::class)
            ->add('image_produit', FileType::class, [
                'label' => 'Image de l\'annonce',
                'required' => false,
                'data_class' => null,
                'empty_data' => 'Aucune image pour ce produit !'
            ])
            ->add('stock_produit', CheckboxType::class)
            ->add('date_depot_produit', DateType::class, [
                'widget' => 'single_text'
            ])
            ->add('prix_produit', NumberType::class)
    }
}
```

Dans votre fichier src/form/ProduitsType.php

Chaque champ est typé à l'aide des classes core type

```
use Symfony\Component\Form\Extension\Core\Type\
```

La méthode new du contrôleur :

```
/**
 * @Route("/new", name="app_produits_new", methods={"GET", "POST"})
 * @param Request $request
 * @param ProduitsRepository $produitsRepository
 * @return Response
 */
public function new(Request $request, ProduitsRepository $produitsRepository): Response
{
    //Instance de entité produits
    $produit = new Produits();
    //Creation du formulaire
    //Creer le formulaire = le methode createForm prend 2 paramètres
    //Le nom de la classe du form builder concerné (php bin/console make:form)
    //AnnonceType spécifie la creation et les paramètres du formulaire
    //En second paramètre : on accède à l'entité annonce et Getters et Setters
    $form = $this->createForm(ProduitsType::class, $produit);
    //Recuperer les champs (input values) du formulaire entré par l'utilisateur
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        //Recupération de la propriété privée de l'image dans l'entité
        $file = $form["image_produit"]->getData();

        //Si la valeur du champ n'est pas de type chaîne de caractères
        if(!is_string($file)){
            //On récupère le nom du fichier uploader
            $fileName = $file->getClientOriginalName();

            //déplacement de la photo = move_uploaded_file($_FILES['userfile']['tmp_name']) en php
            $file->move(
                //Destination du fichier configurer dans le dossier config/services.yaml => parameters
                //images_directory: '%kernel.project_dir%/public/img'
                //Ajouter la ligne a parameters : images_directory: '%kernel.project_dir%/public/img/'
                //En second paramètre = le nom du fichier
                $this->getParameter(name: "images_directory"),
                $fileName
            );
        }
    }
}
```

```

    //Attribution de la photo à l'entité à l'aide des setters
    $produit->setImageProduit($fileName);
    //Notification flash bag
    $this->addFlash( type: 'success', message: 'Votre annonce à bien été ajouté !');
} else{
    //Sinon notif d'erreur
    $this->addFlash( type: 'danger', message: 'Une erreur est survenu durant la création de votre annonce !');
    //redirection vers la page ajouter produits
    return $this->redirect($this->generateUrl( route: 'app_produits_new'));
}

//La requête DQL INSERT INTO
$produitsRepository->add($produit, flush: true);
//Redirection vers la page d'accueil
return $this->redirectToRoute( route: '/produits/', [], status: Response::HTTP_SEE_OTHER);
}

return $this->renderForm( view: 'produits/new', [
    'produit' => $produit,
    'form' => $form,
]);
}

```

Configurer la destination dans votre fichier config/services.yaml

```

parameters:
  images_directory: '%kernel.project_dir%/public/img/'

```

La vue est découpée en 2 parties :

new.html.twig + \_form.html.twig

```

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

2
3  {% block title %}Ajouter Produits{% endblock %}

4
5  {% block body %}
6      <div class="container box">
7          <h1 class="title is-1 is-info">Ajouter un produit</h1>
8
9          {{ include('produits/_form.html.twig') }}
10
11         <a class="button is-info mt-3" href="{{ path('/produits/') }}>ANNULER</a>
12     </div>
13
14  {% endblock %}
15

```

```

{{ form_start(form) }}
{{ form_widget(form) }}
<button class="button is-success mt-3">{{ button_label|default('AJOUTER') }}</button>
{{ form_end(form) }}

```

Résultat :

## Ajouter un produit

**Nom produit**

**Description produit**

**Image de l'annonce**

Aucun fichier choisi

Stock produit

**Date depot produit**

**Prix produit**

## Z-6 – EDITER UN PRODUITS

Mettre à jour le contrôleur pour l'upload de photo :

```
/*
 * @Route("/{id}/edit", name="app_produits_edit", methods={"GET", "POST"})
 * @param Request $request
 * @param Produits $produit
 * @param ProduitsRepository $produitsRepository
 * @return Response
 */
public function edit(Request $request, Produits $produit, ProduitsRepository $produitsRepository): Response
{
    //Recup de la photo courante
    //Recup de l'image avec le getter
    $img = $produit->getImageProduit();
    //Creation du formulaire
    //En paramètre on passe Le formulaire ProduitsType et en 2nd l'entité
    $form = $this->createForm(type: ProduitsType::class, $produit);
    //Recuperation des champs (input) du formulaire d'édition
    $form->handleRequest($request);
    //Si le formulaire est soumis et valide
    if ($form->isSubmitted() && $form->isValid()) {

        //Traitement du fichier upload
        $file = $form['image_produit']->getData();

        if(!is_string($file)){
            $fileName = $file->getClientOriginalName();
            $file->move(
                $this->getParameter(name: 'images_directory'),
                $fileName
            );
            $produit->setImageProduit($fileName);
            $this->addFlash(type: 'success', message: 'Votre photo a bien modifiée !');
        }else{
            $produit->setImageProduit($img);
        }

        //DQL UPDATE et sauvegarde
        $produitsRepository->add($produit, flush: true);

        return $this->redirectToRoute(route: '/produits/', [], status: Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm(view: 'produits/edit', [
        'produit' => $produit,
        'form' => $form,
    ]);
}
```

### Editor un produit

Nom produit

Description produit

Image de l'annonce

Aucun fichier choisi

Stock produit

Date dépôt produit

Prix produit

## Z-7 – Le langage DQL (Doctrine Query Language)

Similaire à SQL, il s'applique à l'entité et non sur la base de données, l'utilisation se fait via un Repository (générer à la création de l'entité) à partir de la méthode `createQuery ()`.

```
$syntaxe = $entityManager->createQuery('Requête DQL');
```

```
$resultats = $syntaxe->getResult();
```

La méthode `getResult ()` récupère les résultats de la requête : il existe différentes méthodes

- `getSingleResult ()` = retourne un seul objet (erreur si pas d'objet ou plusieurs)
- `getOneOrNullResult()` = récupère un objet ou une valeur null (erreur si plusieurs objets)
- `GetArrayResult()` = retourne les résultats sous forme de tableaux imbriqué et renvoie `ArrayList` (ce dernier est différent de `Array` PHP, c'est une classe qui inclut la liste des entités et met à disposition un certain nombre de méthodes)
- `GetScalarResult ()` = retourne des valeurs scalaires qui peuvent contenir des données doubles
- `GetOneScalarResult ()` = retourne une seule valeur scalaire

Exemple de requête DQL : récupérer le dernier produit dans Produit Repository :

Pour ne pas écrire de SQL en dur on utilise les méthodes de Doctrine Symfony `createQueryBuilder ()`

Dans le fichier `ProduitRepository.php` : on ajoute une méthode `getDernierProduit()`

On utilise la méthode `createQueryBuilder` qui prend un alias en paramètre

On effectue un tri décroissant + un nombre de résultat + le query de PDO + retourner un résultat ou rien

Enfin on retourne notre variable `$dernierproduit` ;

```

//Mettre un produit en vedette custom DQL

public function getDernierProduit()
{
    //Ici on crée une variable qui appelle la méthode createQueryBuilder de Doctrine et prend un alias en paramètre
    //De cette manière pas besoin d'écrire de SQL classique
    //A noter que PHP permet de chaîner les méthodes
    $dernierProduit = $this->createQueryBuilder( alias: 'p' )
        //On utilise l'alias p pour récupérer id et trier par ordre décroissant
        ->orderBy( sort: 'p.id', order: 'DESC' )
        //Un seul élément
        ->setMaxResults( maxResults: 1 )
        //Parcours des résultats
        ->getQuery()
        //getOneOrNullResult() = récupère un objet ou une valeur null (erreur si plusieurs objets)
        ->getOneOrNullResult();

    //retourne le résultat de ma requête
    return $dernierProduit;
}

```

On met à jour le contrôleur :

La méthode getDernierProduits() est appelée en fin de méthode

```

public function index(ProduitsRepository $produitsRepository, PaginatorInterface $paginator, Request $request): Response
{
    //Appel du service PaginatorInterface en paramètre
    //Appel de la méthode paginate + paramètres
    $pagination = $paginator->paginate(
        //On récupère tous les articles
        $produitsRepository->findAll(),
        //On liste par entier (knp_paginator.yaml) on définit la clé dans url, par défaut ma page=1 + nombre d'article à afficher (ici 2)
        $request->query->getInt( key: 'page', default: 1 ), limit: 3
    );

    return $this->render( view: 'produits/index', [
        'controller_name' => 'ProduitsController',
        'produits' => $produitsRepository->findAll(),
        'dernierProduit' => $produitsRepository->getDernierProduit(),
        'pagination' => $pagination,
    ]);
}

```

Et la vue : on utilise la clé dernierProduit dans la vue twig

`{{ dernierProduit.nomProduit}}`

```

<div class="alert alert-info columns">
  <div class="column is-4">
    <h3 class="title is-3 has-text-info">SOLDE DU MOIS</h3>
    <h4 class="title is-4 has-text-warning">
      20% de réduction sur le produit : {{ dernierProduit.nomProduit }}</h4>
    </div>
    <div class="card">
      <div class="card-image">
        <figure class="image is-4by3">
          
        </figure>
      </div>
      <div class="card-content">
        <div class="media">
          <div class="media-left">
            <figure class="image is-48x48">
              
            </figure>
          </div>
          <div class="media-content">
            <p class="title is-4">{{ dernierProduit.nomProduit }}</p>
            <p class="subtitle is-6">PRIX : {{ dernierProduit.prixProduit }} € </p>
          </div>
        </div>
        <div class="content">
          <div class="card-action">
            <!-- Ici chemin + route + id de chaque produit -->
            <a href="{{ path('/produits/{id}', {'id': dernierProduit.id}) }}" class="button is-info">Détails du produit</a>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

## SOLDE DU MOIS

Résultat :

20% de réduction sur le produit :  
Canapé



 Canapé  
PRIX : 458.25 €

[Détails du produit](#)

1 2 3 4 5 ... 7

Ajouter un produit

## Z-8 - UN SYSTEME D'INSCRIPTION ET DE CONNEXION SECURISÉ AVEC DES ROLES

3 étapes :

### A – ETAPES 1 : créer la table utilisateurs :

<https://symfony.com/doc/current/security.html#the-user>

On utilise le security-bundle + symfony Flex

**php bin/console make:user**

Répondre aux questions :

- Nom de la table (entité) = User
- Stocker les données en base de données = YES
- Le paramètre de connexion (email, username ou uuid) = email
- haché le mot de passe = YES
- Réalisé les étapes de migration : **php bin/console make:migration**
- **php bin/console doctrine:migrations:migrate**
- Vérifié sur phpMyAdmin que votre table User est bien créée
- On constate que la table générée possède un email + mot de passe + rôle

The screenshot shows the phpMyAdmin interface with the following details:

- Resultat :** MySQL a retourné un résultat vide (c'est à dire aucune ligne). (traitement en 0,0008 seconde(s).)
- SQL Query:** SELECT \* FROM `user`
- Opérations :** Profilage [ Éditer en ligne ] [ Éditer ] [ Expliquer SQL ] [ Créer le code source PHP ] [ Actualiser ]
- Table Headers:** id, email, roles, password
- Operations on Results:** Opérations sur les résultats de la requête
- Buttons:** Crée une vue

### B- ETAPES 2 : générer un formulaire et système d'inscription

- A ce stade il est possible de générer un formulaire d'inscription lié à l'entité User
- **php bin/console make :registration-form**
- Répondre aux questions
- Ajouter à User un paramètre @UniqueEntity pour éviter la duplication d'email = YES
- Envoyer un email pour valider l'inscription = NO
- Connecter utilisateur après inscription = NO
- Quelle route pour la redirection après inscription (page d'accueil) = 0
- La console affiche les fichiers créés et le statut

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

Success!
```

- Entité User a été mise à jour
- Un fichier formulaire a été créer
- Un contrôleur + méthode et route a été créer
- Une vue pour afficher le formulaire d'inscription

LISTER VOS ROUTES DANS LA CONSOLE

**php bin/console debug:router**

Name	Method	Scheme	Host	Path
app_produits_index	GET	ANY	ANY	/produits/
app_produits_new	GET POST	ANY	ANY	/produits/new
app_produits_show	GET	ANY	ANY	/produits/{id}
app_produits_edit	GET POST	ANY	ANY	/produits/{id}/edit
app_produits_delete	ANY	ANY	ANY	/produits/supprimer-produit/{id}
app_register	ANY	ANY	ANY	/register
_preview_error	ANY	ANY	ANY	/_error/{code}.{_format}
_wdt	ANY	ANY	ANY	/_wdt/{token}
_profiler_home	ANY	ANY	ANY	/_profiler/
_profiler_search	ANY	ANY	ANY	/_profiler/search
_profiler_search_bar	ANY	ANY	ANY	/_profiler/search_bar
_profiler_phpinfo	ANY	ANY	ANY	/_profiler/phpinfo
_profiler_search_results	ANY	ANY	ANY	/_profiler/{token}/search/results
_profiler_open_file	ANY	ANY	ANY	/_profiler/open
_profiler	ANY	ANY	ANY	/_profiler/{token}
_profiler_router	ANY	ANY	ANY	/_profiler/{token}/router
_profiler_exception	ANY	ANY	ANY	/_profiler/{token}/exception
_profiler_exception_css	ANY	ANY	ANY	/_profiler/{token}/exception.css

Rendez-vous sur la page : localhost:8000/votre projet /register

Mise à jour de la navbar :

```
<div class="navbar-end">
    <div class="navbar-item">
        <div class="buttons">
            <a href="{{ path('app_register') }}" class="button is-primary">
                <strong>S'INSCRIRE</strong>
            </a>
            <a class="button is-light">
                CONNEXION
            </a>
        </div>
    </div>
</div>
```

Traduire la vue register.html.twig

```

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

{% block title %}Register{% endblock %}

{% block body %}
    <div class="container box mt-3">
        <h1 class="title is-3 has-text-warning">INSCRIPTION</h1>

        {{ form_start(registrationForm) }}
        {{ form_row(registrationForm.email) }}
        {{ form_row(registrationForm.plainPassword, {
            label: 'Password'
        }) }}
        {{ form_row(registrationForm.accepter) }}

        <button type="submit" class="button is-success">S'INSCRIRE</button>
        {{ form_end(registrationForm) }}
    </div>
{% endblock %}

```

Par défaut dans l'entité le rôle de chaque utilisateur est ROLE\_USER

Décommenter les options access\_control dans votre fichier config/packages/securuty/yaml

```

# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }

```

### C- ETAPES 3 : généré la connexion sécurisée

- [https://symfony.com/doc/5.2/security/form\\_login\\_setup.html](https://symfony.com/doc/5.2/security/form_login_setup.html)

- Suivez les instructions
- **php bin/console make:auth**
- Suivez les instructions et répondez aux questions
- Style de connexion = 1 (Login form Authenticator)
- Le nom de la classe = AppCustomAuthenticator
- Le nom du contrôleur = SecurityController

- Générer un système de déconnexion (logout) = yes
- Il faut maintenant gérer la redirection une fois connecter + le rendu du formulaire
- Dans votre dossier src/security/AppCustomAuthenticator.php
- Modifier la redirection une fois connecter

```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }

    // For example:
    return new RedirectResponse($this->urlGenerator->generate(['name' => 'app_produits_index']));
    //throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
}

protected function getLoginUrl(Request $request): string
{
    return $this->urlGenerator->generate(['name' => self::LOGIN_ROUTE]);
}
```

- Mise à jour du menu :

```
<a href="{{ path('app_login') }}" class="button is-light">
    CONNEXION
</a>
```

- Sur la vue : templates/security/login.html.twig

```

❶  {% extends 'base.html.twig' %}

❷  {% block title %}CONNEXION{% endblock %}

❸  {% block body %}
    <form method="post">
        {% if error %}
            <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
        {% endif %}

        {% if app.user %}
            <div class="mb-3">
                Vous êtes connecter en tant que : {{ app.user.userIdentifier }}, <a class="button is-danger" href="{{ path('/logout') }}>DECONNEXION</a>
            </div>
        {% endif %}

        <div class="container box mt-3">
            <h2 class="title is-2 has-text-info">CONNEXION</h2>
            <h3 class="title is-3 has-text-warning">Merci de renter votre email et mot de passe</h3>

            <div class="field">
                <label for="inputEmail">Email</label>
                <div class="control">
                    <input type="email" value="{{ last_username }}" name="email" id="inputEmail" class="input is-danger" autocomplete="email" required autofocus>
                </div>
            </div>

            <div class="field">
                <label for="inputPassword">Password</label>
                <div class="control">
                    <input type="password" name="password" id="inputPassword" class="input is-danger" autocomplete="current-password" required>
                </div>
            </div>

            <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}>

            <button class="button is-info" type="submit">
                CONNEXION
            </button>
        </div>
    </form>
    {% endblock %}

```

- Tester votre connexion :

**CONNEXION**

Merci de renter votre email et mot de passe

Email

Password

CONNEXION

- Vérifié le statut dans le profiler



Lors de la génération du composant Authenticator, ce dernier a généré un système de déconnexion (**php bin/console make:auth**)

Il faut désormais spécifier à la configuration sécurité vers quelle route rediriger l'utilisateur

- Rendez-vous dans votre fichier config/packages/security.yaml

```

firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        lazy: true
        provider: app_user_provider
        custom_authenticator: App\Security\AppCustomAuthenticator
        logout:
            path: app_logout
            # ou rediriger apres déconnexion
            target: app_login

```

L'élément target cible la route de redirection après déconnexion : ici on redirige vers la page de connexion

RAPPEL : pour visualiser les routes dans la console

**php bin/console debug :router**

MODIFIER LA VUE : menu.html.twig

- L'objectif est de modifiée les boutons en fonction du rôle des utilisateur pour rappel les rôles sont les suivants :
- ROLE\_USER, ROLE\_ADMIN, ROLE\_SUPER\_ADMIN

```

<div class="navbar-end">
    <div class="navbar-item">
        <div class="buttons">

            {% if is_granted('ROLE_ADMIN') %}
                <a class="button is-success" href="{{ path('admin') }}"
                   ADMINISTRATION
                </a>

                <a href="{{ path('app_logout') }}" class="button is-danger">
                   DÉCONNEXION
                </a>
            {% else %}
                <a href="{{ path('app_login') }}" class="button is-light">
                   CONNEXION
                </a>

                <a href="{{ path('app_register') }}" class="button is-primary">
                   <strong>S'INSCRIRE</strong>
                </a>
            {% endif %}

        </div>
    </div>
</div>

```

## Z-9 – Les relations entre les entités

Avec SQL - DQL ou QueryBuilder on peut faire des jointures entre les tables, on fait donc appel à des clés étrangères et des tables intermédiaires

4 Sortes de jointures :

OneToOne :

- Un enregistrement de la table propriétaire ne peut être lié qu'à un seul enregistrement de la table secondaire (ex : Un produit => une seule référence (n° facture))

OneToMany :

- Un enregistrement de la table propriétaire est lié à plusieurs enregistrements de la table secondaire mais pas réciproquement (ex : Un ordinateur => PC, MAC, Pc portable, etc...)

ManyToOne :

- Autant de relation que l'on souhaite entre la table propriétaire et la table secondaire et réciproquement (ex : Spaghetti => Butonni, Panzani, Barilla, top budget, Casino, etc...)

ManyToMany :

- Les relations sont complètes, autant de relation que l'on souhaite entre les deux tables (ex : Produits => marques)

## Z-10 – UNE RELATION ONE TO ONE

- Pour illustrer une relation One to One nous allons créer une table

(entité) Références

- LES CARDINALITES

- Un produit possède une et une seule référence et une référence appartient à un seul produit

- Dans la console :

**php bin/console make:entity**

- Nom de l'entité = Références

- Un champ numero\_reference de type Integer

- Réaliser la migration : **php bin/console make:migration**

- Valider la migration : **php bin/console doctrine:migrations:migrate**

- Vérifier sur PhpMyAdmin la création de la table

METTRE A JOUR LA TABLE (entité) PRODUITS

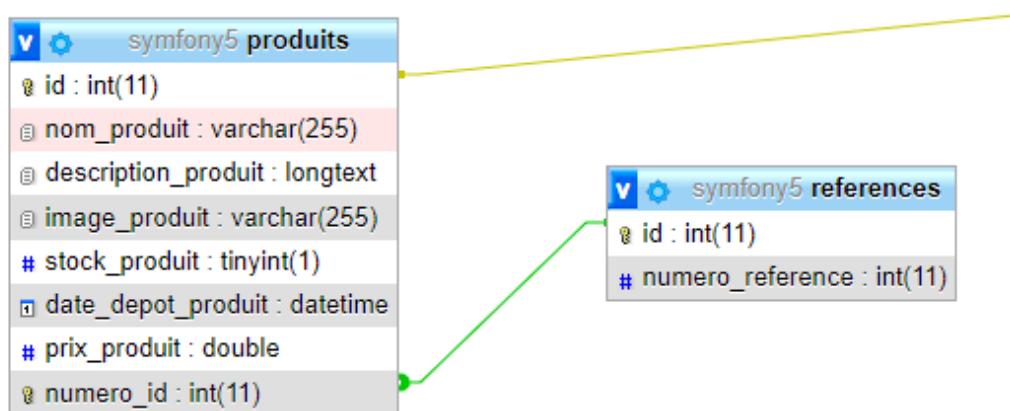
**php bin/console make:entity**

- Entrer le nom de l'entité existante Produits
- Ajouter un champ numéro
- Le type est relation

- Cette relation est lié à la classe Références
- Quelle type de relation ? = OneToOne
- La jointure Produits.numero peut-il être nul = no
- Voulez-vous ajouter une propriété à Reference pour accéder au Produits = no
- Exécuter de nouveau la migration de l'entité Produits :

**php bin/console make:migration**

- Valider la migration : **php bin/console doctrine:migrations:migrate**
- Vérifié l'opération dans PhpMyAdmin :



- Afficher la clé étrangère dans la vue
- Cette étape consiste à appeler votre alias de la boucle for + le getter de la clé étrangère + le champ de la table référence

```

{%
if produit.numero is not null %}
<!--alias + champ Produits (cle étrangère) + champ Table Reference (getter)-->
<p class="has-text-info">Reference du produit (clé étrangère numero) : {{ produit.numero.numeroReference }}</p>
{%
endif %}

```

- Dans l'entité Produits mis à jour : on constate des annotation réciproque :

Produits.php

```

/**
 * @ORM\OneToOne(targetEntity=References::class, cascade={"persist"})
 * @ORM\JoinColumn(nullable=true)
 * @return int|null
 */
private $numero;

```

Dans la vue index.html.twig

```

{%
if produit.numero is not null %}
<!--alias + champ Produits (clé étrangère) + champ Table Reference (getter)-->
<p class="has-text-info">Reference du produit (clé étrangère numero) : {{ produit.numero.numeroReference }}</p>
{%
endif %}

```

Le résultat :



Pour ajouter un produit : il faut mettre à jour notre fichier ProduitsType

On utilise ici des formulaires imbriquer (c'est à dire un formulaire pour la référence est imbriqué dans le formulaire produits)

- Dans votre dossier src/form : créer une classe ReferencesType.php

```

<?php

namespace App\Form;

use App\Entity\References;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\NumberType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ReferencesType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('numeroReference', NumberType::class, [
                'label' => 'Référence de l\'annonce'
            ]);
    }

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

```

- Ensuite ajouter ce champ à src/form/ProduitsType.php

```

//ceci est un formulaire imbriqué
//On appelle form/ReferencesType form

->add('numero', ReferencesType::class, [
    'required' => true
])

```

Lors de l'ajout d'un produit : nous avons un nouveau champ

Numero	<input type="text"/>
Référence de l'annonce	<input type="text"/>

## Z-11 – Une relation Many To Many et les tables intermédiaires

- Ajouter une nouvelle entité Distributeur :

**php bin/console make:entity**

- Ajouter une propriété nom\_distributeur : string varchar 255
- la migration : **php bin/console make:migration**
- Puis : **php bin/console doctrine:migrations:migrate**
- Vérifié le résultat dans phpMyAdmin
- Modifier l'entité Produits (on ajoute une clé étrangère distributeur)

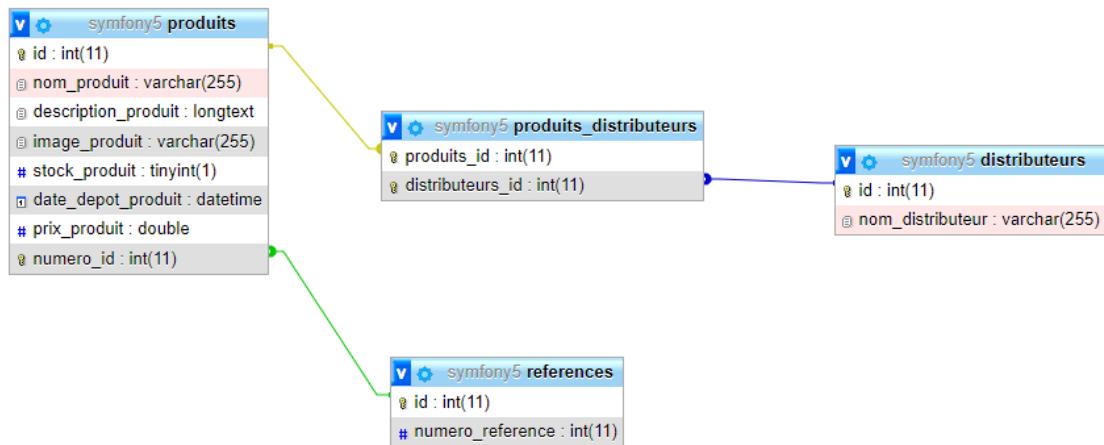
**php bin/console make:entity Produits**

- Ajouter le champs distributeur : type relation
- On lie l'entité Produits à l'entité Distributeurs
- Le type de relation est ManyToMany
- Ajouter une propriété a Distributeurs pour accéder aux Produits = YES

**php bin/console make:migration**

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

- Ces opérations ont pour effet de créer une table intermédiaire entre Produits et Distributeurs



Pour ajouter un produit, il faut évidemment mettre à jour votre formulaire ProduitsType :

- Le type de champs est EntityType
- On ajoute la jointure ManyToMany via la clé 'class' => Entité Distributeurs
- Ce champs est un select => option Multiple HTML

```

->add( child: 'distributeurs', type: EntityType::class,
      'class' => Distributeurs::class,
      'choice_label' => 'nomDistributeur',
      'label' => 'Selectionnez un ou plusieurs distributeurs',
      'multiple' => true
)

```

- Au niveau des entités les annotations utilisent une relation (jointure) réciproque

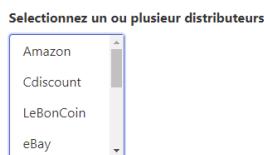
Produits.php

```
/**  
 * @ORM\ManyToMany(targetEntity=Distributeurs::class, inverseBy="produits")  
 */  
private $distributeurs;
```

- Idem pour l'entité Distributeurs.php

```
/**  
 * @ORM\ManyToMany(targetEntity=Produits::class, mappedBy="distributeurs")  
 */  
private $produits;
```

Pour la vue :



Ainsi la table produits\_distributeurs (la table intermédiaire entre produits et distributeurs) s'occupe de faire les jointures entre les 2 clés primaires

Dans la vue index.html.twig

```
<!--Plusieur distributeur donc boucle for + champs table Distributeurs = alias + getter-->  
{% for distributeur in produits.distributeurs %}  
    <p class="has-text-danger">Distributeur(s) :{{ distributeur.nomDistributeur }}</p>  
{% endfor %}
```

## Z-12 – Une relation One To Many et Many To One

- On ajoute à nos produits une catégorie
- Créer une entité Catégories avec les champs suivants :

**php bin/console make :entity Categories**

- Ajouter le champ nom\_categorie string 255

**php bin/console make:migration**

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

- Vérifié le résultat dans PhpMyAdmin et ajouter quelques catégories

Affichage des lignes 0 - 11 (total de 12, traitement en 0.0007 seconde(s).)																															
SELECT * FROM `categories`																															
<input type="checkbox"/> Profilage [ Éditer en ligne ] [ Éditer ] [ Expliquer SQL ] [ Créer le code source PHP ] [ Actualiser ]																															
<input type="checkbox"/> Tout afficher	Nombre de lignes :	25	Filtrer les lignes:	Chercher dans cette table	Trier par clé : Aucun(e)																										
<input checked="" type="checkbox"/> Options																															
<table border="1"> <thead> <tr> <th style="text-align: center;">id</th><th style="text-align: center;">nom_categorie</th></tr> </thead> <tbody> <tr><td style="text-align: center;">1</td><td>Vacances</td></tr> <tr><td style="text-align: center;">2</td><td>Emploi</td></tr> <tr><td style="text-align: center;">3</td><td>Véhicules</td></tr> <tr><td style="text-align: center;">4</td><td>Immobilier</td></tr> <tr><td style="text-align: center;">5</td><td>Mode</td></tr> <tr><td style="text-align: center;">6</td><td>Maison</td></tr> <tr><td style="text-align: center;">7</td><td>Multimedia</td></tr> <tr><td style="text-align: center;">8</td><td>Loisirs</td></tr> <tr><td style="text-align: center;">9</td><td>Animaux</td></tr> <tr><td style="text-align: center;">10</td><td>Materiel Pro</td></tr> <tr><td style="text-align: center;">11</td><td>Services</td></tr> <tr><td style="text-align: center;">12</td><td>Divers</td></tr> </tbody> </table>						id	nom_categorie	1	Vacances	2	Emploi	3	Véhicules	4	Immobilier	5	Mode	6	Maison	7	Multimedia	8	Loisirs	9	Animaux	10	Materiel Pro	11	Services	12	Divers
id	nom_categorie																														
1	Vacances																														
2	Emploi																														
3	Véhicules																														
4	Immobilier																														
5	Mode																														
6	Maison																														
7	Multimedia																														
8	Loisirs																														
9	Animaux																														
10	Materiel Pro																														
11	Services																														
12	Divers																														
<input type="checkbox"/> Tout cocher      Avec la sélection : <input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer <input type="checkbox"/> Exporter																															

- Ajouter une clé étrangère à l'entité Produits de type relation OneToMany

## LES CARDINALITES

Plusieurs produits ont Une et Une seule Catégories (ManyToOne) et réciproquement Une Catégorie appartient à UN ou PLUISEUR Produits (OneToMany)

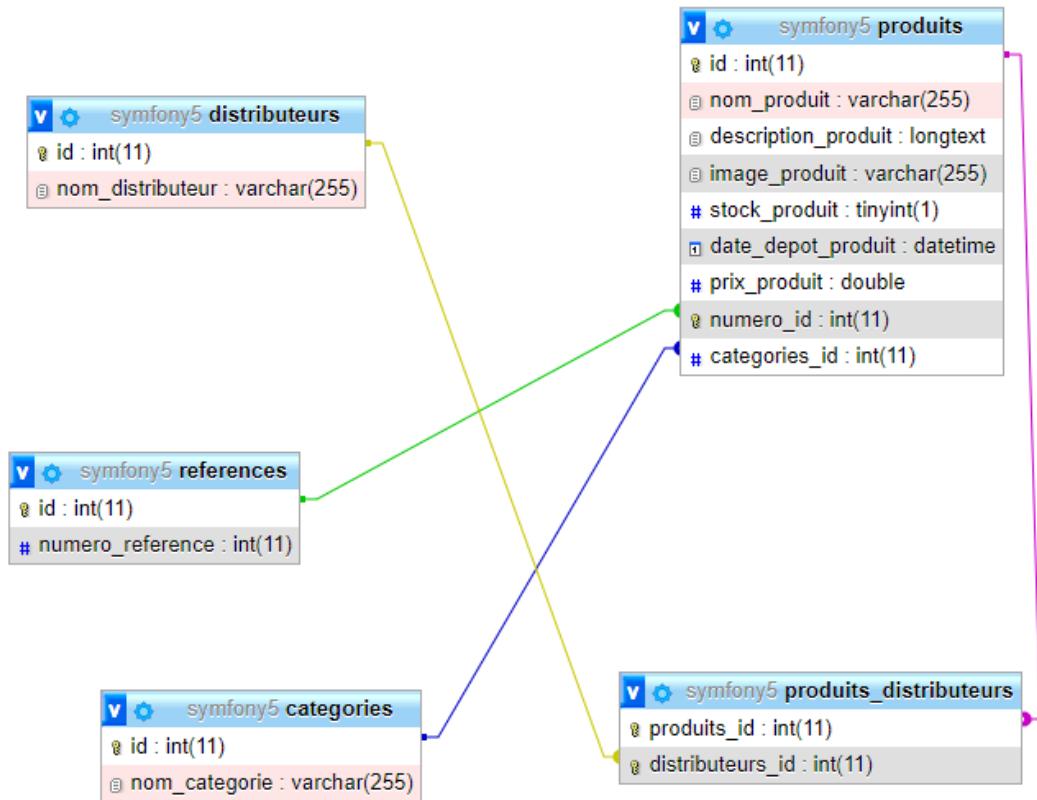
**php bin/console make:entity Produits**

- Ajouter un champ catégories
- Type relations
- Lie à l'entité Catégories
- Relation ManyToOne (la réciprocité est automatique)
- Produits.categorie peut être nul = no
- Catégories peut accéder à produit = yes
- Categories.produits
- Supprimer automatiquement les orphelins = no

**php bin/console make:migration**

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

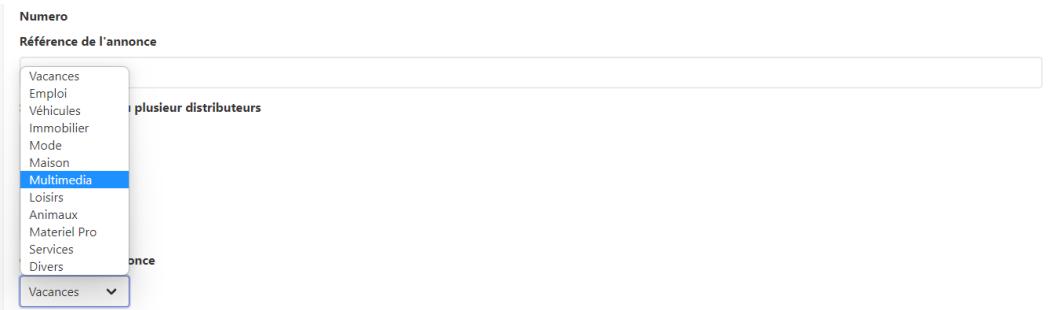
- Vérifié le résultat sur PhpMyAdmin :



Mettre à jour votre formulaire ProduitsType.php :

```
->add('categories', type: EntityType::class, [
    'class' => Categories::class,
    'choice_label' => 'nomCategorie',
    'label' => 'Catégorie de l\'annonce',
    'required' => true
])
```

La vue :



Dans votre page Twig :

```
<!--alias + champ Produits (clé étrangère) + champ Table Categories (getter)-->
<p class="has-text-warning">Catégorie : (clé étrangère ManyToOne & OneToMany) : {{ produit.categories.nomCategorie }}</p>
```

## Z-13 – Le Reverse Engineering :

Si vous utilisez une base de données déjà existante il sera fastidieux de créer à la main toutes les entités, pour cela Symfony a prévu un cas :

Créer les entités automatiquement :

La commande suivante : `php bin/console doctrine :mapping :import "App\Entity" annotation --path=src/Entity/Reverse`

Dans le dossier src/Entity ce créera un dossier Reverse qui contiendra toutes les entités générées.

Les jointures sont détectées

Il faudra régénérer les entités : `php bin/console make :entity --regenerate App`

## Z-14 - ADMINISTRATION : LE BUNDLE EASY ADMIN

- **Erreur ! Référence de lien hypertexte non valide.**

- Le bundle EasyAdmin créer des backends d'administration pour votre site web rapidement
- il est important de comprendre que le bundle ne faut pas tous et que certaines étapes devront être réalisées à la main
- Installer le bundle

`composer require easycorp/easyadmin-bundle`

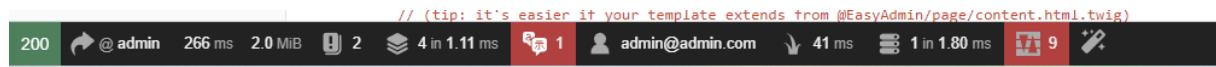
- Générer un tableau de bord pour les administrateurs

```
php bin/console make:admin:dashboard
```

- Le nom du Controller = DashboardController
- Le répertoire = src/Controller/Admin
- Créer un utilisateur à l'aide de votre formulaire d'inscription
- Dans PhpMyAdmin : modifier son rôle
- [“ ROLE\_ADMIN ”] = Attention ici le format est du Json, il faut impérativement mettre des guillemet double
- Dans votre fichier config/packages/security.yaml : les routes /admin demande le rôle Admin pour être accessibles

```
# Les routes qui commence par /admin nécessite le rôle ADMIN
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }
```

- Connectez-vous avec compte administrateur et vérifier le statut



- Rendez-vous sur l'URL suivante : localhost:8000/admin

- Nous allons générer un CRUD Admin pour chaque table

```
php bin/console make:admin:crud
```

- Choisissez l'entité sur lequel vous souhaitez réaliser le CRUD
- Le dossier est src/Controller/Admin/ crud de chaque entité
- Idem pour le namespace

Rendez-vous dans votre fichier src/Controller/Admin/DashboardController.php

Dans la méthode configureMenuItems() : Créer un tableau et appeler vos entités

```
public function configureMenuItems(): iterable
{
    return [
        MenuItem::linkToDashboard( label: 'Tableau de bord', icon: 'fa fa-home'),

        MenuItem::section( label: 'Produits'),
        MenuItem::linkToCrud( label: 'Produits', icon: 'fa fa-user', entityFqcn: Produits::class),

        MenuItem::section( label: 'Catégories'),
        MenuItem::linkToCrud( label: 'Catégories', icon: 'fa fa-user', entityFqcn: Categories::class),

        MenuItem::section( label: 'Distributeurs'),
        MenuItem::linkToCrud( label: 'Distributeurs', icon: 'fa fa-user', entityFqcn: Distributeurs::class),

        MenuItem::section( label: 'Références'),
        MenuItem::linkToCrud( label: 'Références', icon: 'fa fa-user', entityFqcn: References::class),

        MenuItem::section( label: 'Utilisateurs'),
        MenuItem::linkToCrud( label: 'Utilisateurs', icon: 'fa fa-user', entityFqcn: User::class),
    ];
}
```

Par défaut la page d'accueil doit être modifier pour laisser apparaitre votre table Produits

```
/*
 * @Route("/admin", name="admin")
 */
public function index(): Response
{
    $routeBuilder = $this->get(AdminUrlGenerator::class);

    return $this->redirect($routeBuilder->setController(crudControllerFqcn: ProduitsCrudController::class)->generateUrl());
}
```

On appelle la classe AdminUrlGenerator et on lui stipule que la page d'accueil est votre contrôleur Produits

Résultat :

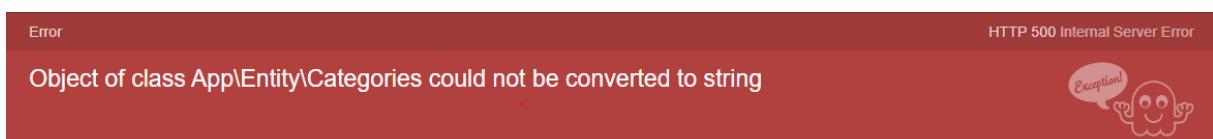
ID	Nom Produit	Image Produit	Stock Produit	Date Dépot Produit	Prix Produit
23	Chariot à roulette	chariot.jpg	1	8 déc. 1973 à 00:00:00	573.302
26	iPhone	iphone.jpg	1	17 juin 2022 à 00:00:00	458.25
27	Lego	lego.jpg	1	11 juin 2022 à 00:00:00	50.25

## PERSONALISER EASY ADMIN ET LE CRUD PRODUITS

- Par défaut easy admin n'a pas inclus de champ input file pour télécharger les photos des produits,
- Rendez-vous dans votre dossier  
src/Controller/Admin/ProduitsCrudController.php
- Décommenter la méthode configureFields
- Modifier chaque champ pour customiser le tableau de bord

```
public function configureFields(string $pageName): iterable
{
    return [
        IntegerField::new('id', 'label: 'ID')->onlyOnIndex(),
        TextField::new('nomProduit', 'label: "Nom du produit"),
        TextEditorField::new('descriptionProduit', 'label: 'Description du produit'),
        NumberField::new('prixProduit', 'label: 'prix du produit'),
        ImageField::new('imageproduit')
            ->setBasePath('path: '/img')
            ->setUploadDir('uploadDirPath: 'public/img/')
            ->setFormType('formTypeFqn: FileUploadType :: class')
            ->setRequired('isRequired: true),
        BooleanField::new('stockProduit', 'label: 'Produit en stock'),
        DateTimeField::new('dateDepotProduit', 'label: 'date de depot'),
        AssociationField::new('categories', 'label: 'Catégorie du produit'),
        AssociationField::new('distributeurs', 'label: 'Liste des distributeurs'),
        IntegerField::new('numero', 'label: 'Référence de l\'annonce')
            ->setFormType('formTypeFqn: ReferencesType :: class')
    ];
}
```

Si vous rencontrez l'erreur suivante :



Dans votre entité catégories ajoute la méthode magique \_\_toString() pour convertir votre clé étrangère en chaîne de caractères (idem pour les distributeurs)

```
public function __toString()
{
    return $this->nom_distributeur;
```

Puis :

```
public function __toString(){
    return $this->nom_categorie;
}
```

Résultat :

**Créer "Produits"**

Nom du produit

Description du produit

prix du produit

Imageproduit\*

Imageproduit

Produit en stock

Date de dépôt

Catégorie du produit\*

Vacances

Liste des distributeurs

Référence de l'annonce

Référence de l'annonce

Créer et ajouter un nouvel élément **Créer**

Pour la table Crud Utilisateur, nous allons ajouter les rôles :

Le principe est le même :

- Rendez-vous dans votre fichier  
src/Controller/Admin/UserCrudController.php
- Décommenter la méthode configureFields

- Ajouter le code suivant :

```

    public function configureFields(string $pageName): iterable
{
    return [
        IntegerField::new( propertyName: 'id', label: 'ID')->onlyOnIndex(),
        EmailField::new( propertyName: 'email', label: 'Email'),
        TextField::new( propertyName: 'password', label: 'Mot de passe'),
        ChoiceField::new( propertyName: 'roles', label: 'Role de l\'utilisateur')
            ->setChoices([
                'ROLE_USER' => 'ROLE_USER',
                'ROLE_ADMIN' => 'ROLE_ADMIN',
                'ROLE_SUPER_ADMIN' => 'ROLE_SUPER_ADMIN'
            ])
            ->allowMultipleChoices( allow: true)
    ];
}

```

Résultat :

User					<a href="#">Créer User</a>
	ID	Email	Mot de passe	Role de l'utilisateur	
<input type="checkbox"/>	1	michael@gmail.com	\$2y\$13\$ikNbAY6lRzZ7FhkHMAQXL.SvM/QZJTV/9FkcAmTHh8Fr9SOnRexy	ROLE_USER	...
<input type="checkbox"/>	2	admin@admin.com	\$2y\$13\$HWLdt9k0ea2DkXwd8DjN.5A3HXXFlLe0ejlQ5Drqn8n47jSLDR8.	ROLE_ADMIN, ROLE_USER	...

2 résultats [Précédent](#) 1 [Suivant >](#)

Votre espace d'administration est maintenant opérationnelle

LES PLUS :

Cacher des éléments en fonction des rôles des utilisateurs :

Si le l'internaute n'est pas connecté en tant qu'administrateur : on cache les accès au CRUD via le contrôleur :

Un exemple avec la méthode new pour ajouter un produit

```

    /**
 * @Route("/new", name="app_produits_new", methods={"GET", "POST"})
 * @Security("is_granted('ROLE_ADMIN')") //cache le bouton ajouter si on est pas Admin
 * @param Request $request
 * @param ProduitsRepository $produitsRepository
 * @return Response
 */
public function new(Request $request, ProduitsRepository $produitsRepository): Response
{

```

Ici l'annotation `@Security("is_granted('ROLE_ADMIN')")` interdit l'accès à cette méthode si l'utilisateur n'a pas le rôle ADMIN définis lors de la connexion et le fichier security.yaml

La même action avec twig

```

{%- if is_granted('ROLE_ADMIN') %}

<div class="mt-3">
    <a href="{{ path('/produits/new') }}" class="button is-light mt-3">Ajouter un produit</a>
</div>
{%- endif %}

```

Et les autres boutons

```

{%- if is_granted('ROLE_ADMIN') %}

<div class="content">
    <div class="card-action">
        <!-- Ici chemin + route + id de chaque produit -->
        <a href="{{ path('/produits/{id}', {'id': produit.id}) }}" class="button is-info">Détails du produit</a>

        <!-- Supprimer annonces -->
        <div style="...">
            {{ include('produits/_delete_form.html.twig') }}
        </div>

        <a href="{{ path('/produits/{id}/edit', {'id': produit.id}) }}" class="button is-warning mt-4">Editer le produit</a>
    </div>
</div>
{%- endif %}

```

Résultat :

The screenshot shows a search results page for "Nombre total de produits : 4". There are three items listed:

- Chariot à roulette** (Prix : 573,302 €) - An image of a black folding shopping cart on a wooden deck.
- Iphone** (Prix : 458,25 €) - An image of an iPhone next to its box.
- Lego** (Prix : 50,25 €) - An image of a large pile of colorful LEGO bricks.

Each item has a "Plus d'info" button at the bottom right. Navigation buttons for page 1 and 2 are at the top left, and "Précédent" and "Suivant" are at the top right.

## Z-15 – AJOUTER UN TABLEAU DE BORD POUR LES UTILISATEURS

- L'objectif est de créer une relation (jointure) OneToMany entre les utilisateurs (ROLE\_USER) et l'entité produits
- Il va falloir donc ajoutés une clé étrangère à la table Produits en relation avec un Utilisateur

`php bin/console make :entity Produits`

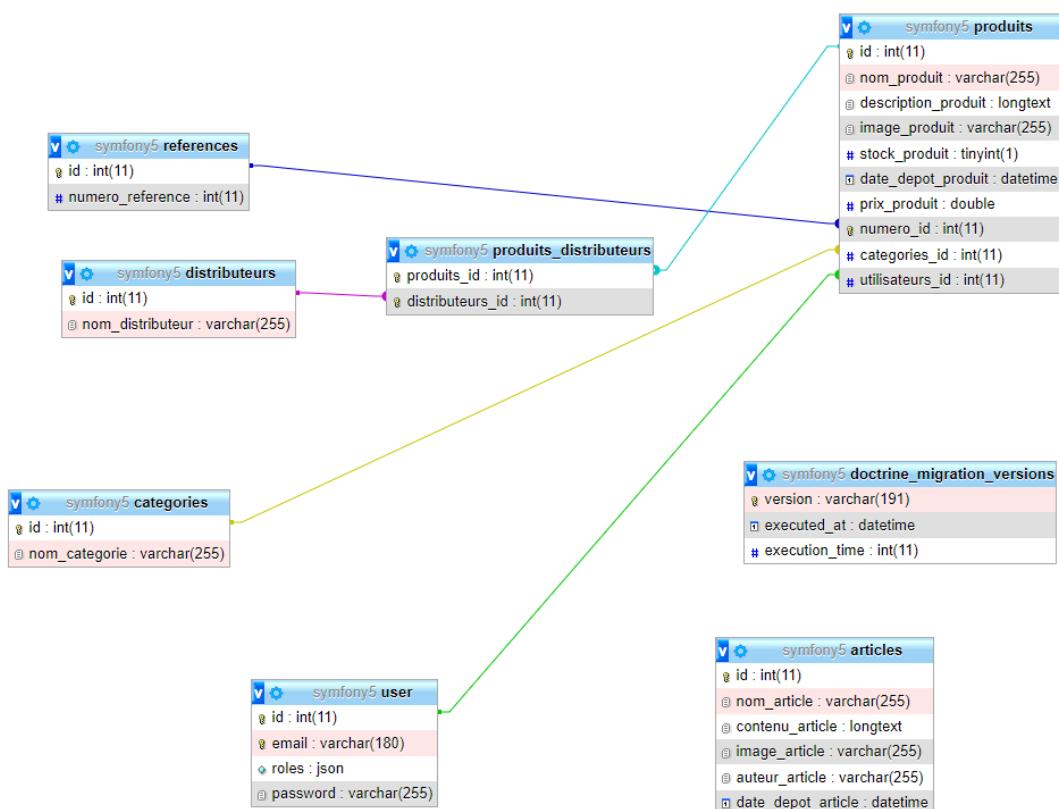
- Ajouter un champs utilisateurs
- Field Type = relation
- Lié à la classe (entité) User
- Type de relation = ManyToOne
- Plusieurs produits sont liés à un et un seul utilisateur
- Est-ce que Produits.utilisateurs peut être nul = NO
- Ajouter une propriété à la classe User pour accéder aux produits = YES
- Le nom du champ dans la classe User = produits
- Activer la suppression orpheline des produits lors de la suppression d'un utilisateur = NO
- Valider le champ

Comme d'habitude effectuer la migration et le flush

**php bin/console make:migration**

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

Vérifié que la jointure a fonction sous PhpMyAdmin :



Dans l'entité Produits :

```

/**
 * @ORM\ManyToMany(targetEntity=User::class, inverseBy="produits")
 * @ORM\JoinColumn(nullable=false)
 */
private $utilisateurs;

```

Et réciproquement dans l'entité User :

```

/**
 * @ORM\OneToMany(targetEntity=Produits::class, mappedBy="utilisateurs")
 */
private $produits;

```

Pour le test ajouter des utilisateurs aux ROLE\_USER

+ Options											
	id	nom_produit	description_produit	image_produit	stock_produit	date_depot_produit	prix_produit	numero_id	categories_id	utilisateurs_id	
<input type="checkbox"/> Copier  Supprimer	23	Chariot a roulette	Chariot en fer a roulette	chariot.jpg	1	1973-12-08 00:00:00	573.302	1	10	1	
<input type="checkbox"/> Copier  Supprimer	26	Iphone	Telephone apple occasion	iphone.jpg	1	2022-06-17 00:00:00	458.25	2	2	4	
<input type="checkbox"/> Copier  Supprimer	27	Lego	Vrac de Lego pour les enfants	lego.jpg	1	2022-06-11 00:00:00	50.25	3	8	3	
<input type="checkbox"/> Copier  Supprimer	33	Test ajout de produit	<div>OK</div>	livre.jpg	1	2022-06-04 10:34:00	4515	4	8	4	

Ainsi Laura possède 2 Produits et Tony un seule

Afficher l'email du vendeur aux produits :

Mettre à jour le code de la vue index.html.twig

```

<p class="has-text-info">Nom du vendeur : {{ produit.utilisateurs.email }}</p>

```

Résultat :



Afficher les produits par utilisateurs :

- Se connecter avec un utilisateur ROLE\_USER
- Ajouter à la vue templates/produits/inex.html.twig le code suivant

```
{% if app.user %}
    <div class="mt-3 text-center">
        <h4 class="title is-3 has-text-danger" style="width: 100%; background-color: #1c1917; padding: 20px">TABLEAU DE BORD : {{ app.user.email }}</h4>
    </div>
{% endif %}
```

- Et une boucle {%- for %} pour afficher le produit par utilisateur

```
!————— ESPACE UTILISATEUR —————
<div class="columns is-multiline mt-3" style="background-color: #0c4a6e">
    {% if app.user %}
        <style>
            #produits-visiteur{
                display: none;
            }
            #solde{
                display: none;
            }
            #produit-counter{
                display: none;
            }
        </style>
        {% for produit in app.user.produits %}
            <div class="column is-4 mt-3">
                <div class="card">
                    <div class="card-image">
                        <figure class="image is-4by3">
                            
                        </figure>
                    </div>
                    <div class="card-content">
                        <div class="media">
                            <div class="media-left">
                                <figure class="image is-48x48">
                                    
                                </figure>
                            </div>
                            <div class="media-content">
                                <p class="title is-3 has-text-info">Nom du vendeur : {{ produit.utilisateurs.email }}</p>
                                <p class="title is-4">{{ produit.nomProduit }}</p>
                                <p class="subtitle is-6">PRIX : {{ produit.prixProduit }} € </p>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        {% endfor %}
    {% endif %}
</div>
```

```

{% if produit.numero is not null %}
    <!--alias + champ Produits (cle étrangère) + champ Table Reference (getter)-->
    <p class="has-text-info">Référence du produit (cle étrangère numero) : {{ produit.numero.numeroReference }}</p>
{% endif %}

<!--alias + champ Produits (cle étrangère) + champ Table Categories (getter)-->
<p class="has-text-warning">Catégorie : (cle étrangère ManyToOne & OneToMany) : {{ produit.categories.nomCategorie }}</p>

<!--Plusieurs distributeur donc boucle for + champs table Distributeurs = alias + getter-->
{% for distributeur in produit.distributeurs %}
    <p class="has-text-danger">Distributeur(s) :{{ distributeur.nomDistributeur }}</p>
{% endfor %}
<div class="content-footer mt-3 text-center">
    <a href="{{ path('app_produits_show', {'id': produit.id}) }}" class="button is-primary">Plus d'info</a>
</div>

</div>
</div>

<div class="content">
    <div class="card-action">
        <!--Ici chemin + route + id de chaque produit-->
        <a href="{{ path('app_produits_show', {'id': produit.id}) }}" class="button is-info">Détails du produit</a>

        <!--Supprimer annonces-->
        <div style="margin-top: 20px">
            {{ include('produits/_delete_form.html.twig') }}
        </div>

        <a href="{{ path('app_produits_edit', {'id': produit.id}) }}" class="button is-warning mt-4">Editer le produit</a>
    </div>
</div>
</div>

```

Résultat :

Laura a bien ses 2 produits :

TABLEAU DE BORD : laura@gmail.com

<b>Nom du vendeur :</b> laura@gmail.com	<b>Nom du vendeur :</b> laura@gmail.com
<b>Iphone</b> PRIX: 458.25 €	<b>Test ajout de produit</b> PRIX: 4515 €
Référence du produit (cle étrangère numero) : 12586 Catégorie : (cle étrangère ManyToOne & OneToMany) : Emploi Distributeur(s) : Amazon Distributeur(s) : LeBonCoin	Référence du produit (cle étrangère numero) : 78458963 Catégorie : (cle étrangère ManyToOne & OneToMany) : Loisirs Distributeur(s) : Cdiscount Distributeur(s) : eBay
<a href="#">Plus d'info</a>	<a href="#">Plus d'info</a>
<a href="#">Détails du produit</a>	<a href="#">Détails du produit</a>
<a href="#">Supprimer</a>	<a href="#">Supprimer</a>
<a href="#">Editer le produit</a>	<a href="#">Editer le produit</a>

Et Tony également :

The screenshot shows a user profile titled "TABLEAU DE BORD : tony@laposte.net". At the top, there are navigation links "1" (selected), "2", "Précédent", and "Suivant". Below the title is a circular image of a pile of various colored LEGO bricks. To the right of the image, there is a large blue area containing a product listing for a "Lego". The listing includes the following details:

- Nom du vendeur :** tony@laposte.net
- Lego**
- PRIX :** 50.25 €
- Reference du produit (clé étrangère numéro) :** 784589
- Catégorie :** (clé étrangère ManyToOne & OneToMany) : Loisirs
- Distributeur(s) :** LeBonCoin
- Distributeur(s) :** eBay

Below the product details are several buttons:  
Plus d'info (in a dark blue box)  
Détails du produit (in a light blue box)  
Supprimer (in a dark red box)  
Editer le produit (in a light gray box)

Il est possible d'ajouter des produits pour chaque utilisateur connecté !

## Z-17 – AJOUTER UNE BARRE DE RECHERCHE PAR FOURCHETTE DE PRIX

La première étape consiste créer une méthode dans ProduitsRepository

On utilise le langage DQL + une fonction queryBuilder() de Symfony

<https://symfony.com/doc/current/doctrine.html#querying-for-objects-the-repository>

Dans src/repository/ProduitsRepository.php

```
//Trier les annonces par prix et mot cle
//Cette méthode est appelée dans RechercherController
public function getMinMaxPrice($prixMin, $prixMax){
    //Creation du queryBuilder + alias de entité Annonce
    $query = $this->createQueryBuilder( alias: 'produits');

    //Si les 2 champs sont remplis
    if($prixMin && $prixMax){
        $query
            ->andWhere('produits.prix_produit BETWEEN :prixmin AND :prixmax')
            ->setParameter( key: 'prixmin', $prixMin)
            ->setParameter( key: 'prixmax', $prixMax);
    }

    return $query->getQuery()->getResult();
}
```

Cette méthode sera appelée dans RechercherController.php

Générer RechechrController.php

**php bin/console make :controller RechercherController**

Cette étape a pour effet de générer un contrôleur et une vue dans le dossier templates/rechercher/index.html.twig

Pour simplifier les tâches nous allons générer le formulaire de recherche directement dans le Controller

(La bonne pratique consiste à générer un formulaire ex :

**php bin/console make :form RechercherType**

Le fichier src/Controller/RechercherController.php

```

class RechercherController extends AbstractController
{
    /**
     * @Route("/rechercher", name="app_rechercher")
     */
    //J'ajout de l'abstrait Request + ProduitsRepository (DQL) + retourne l'abstrait Response
    public function rechercher(Request $request, ProduitsRepository $produitsRepository): Response
    {
        //on definit des variables criteres de recherche
        $prixMin = "";
        $prixMax = "";

        $message = [
            "message" => "Vos resultats : "
        ];
        //On creer un formulaire customiser directement dans le controller a la place de la vue
        //On utilise la methode createFormBuilder
        $formulaireRecherche = $this->createFormBuilder($message)
            //Le champ prix min
            ->add( child: "prixMin", type: NumberType::class,[
                "label" => "Prix minimum du produit",
                "required" => false
            ])
            //le champs prix max
            ->add( child: "prixMax", type: NumberType::class,[
                "label" => "Prix maximum du produit",
                "required" => false
            ])

            //le bouton de recherche
            ->add( child: "rechercher", type: SubmitType::class,[
                "label" => "Rechercher",
                "attr" => ['class' => 'button is-danger']
            ])
            //Ici on utilise la methode getForm() pour finaliser la creation du formulaire
            ->getForm();
        //Recuperation des donnees du champs (<input name=""> soit form[prixMin] form[prixMax] et form[mot] (f12 dans le navigateur))
        $formulaireRecherche->handleRequest($request);

        //Si le formulaire est soumis et est valide
        if($formulaireRecherche->isSubmitted() && $formulaireRecherche->isValid()){
            $resultat = $formulaireRecherche->getData();
            //Debug = var_dump()
            //dd($resultat);
            $prixMin = $resultat['prixMin'];
            $prixMax = $resultat['prixMax'];
        }
        return $this->render( view: 'rechercher/index', [
            'controller_name' => 'RechercherController',
            'formulaireRecherche' => $formulaireRecherche->createView(),
            //Appel de produit repository + custom queryBuilder
            //Cette methode prend 3 parametres definis ci dessus =
            'resultatsRecherche' => $produitsRepository->getMinMaxPrice($prixMin, $prixMax)
        ]);
    }
}

```

Beaucoup d'éléments ici :

On passe en paramètre de la méthode :

- Objet Request
- ProduitsRepository

Dans la fonction :

- Créer 2 variables \$prixMin + \$prixMax
- Un tableau message

- Créer un formulaire avec createFormBuilder()
- Ajouter 3 champs de type NumberType
- Appeler la méthode handleRequest pour récupérer les \$\_POST
- Dans une condition : Si le formulaire est soumis et valide
- Récupérer les valeurs du formulaire
- \$résultat = \$formulaireRecherche->getData();
- Affecter les variables \$prixMin et Max aux \$\_POST
- \$prixMin = \$résultat['prixMin'];  
\$prixMax = \$résultat['prixMax'];
- Appel la vue via la méthode render
- Dans un tableau associatif
- Créer une clé 'résultatRecherche' => appel de la méthode du ProduitsRepository
- 'résultatsRecherche' => \$produitsRepository->getMinMaxPrice(\$prixMin, \$prixMax)

La Vue templates/rechercher/index.html.twig

```

1 ①  {% extends 'base.html.twig' %}

2 ②  {% block title %}SF5 -RECHERCHE-{{ endblock %}

3 ③  {% block body %}
4    <div class="container">
5      <h1 class="text-info text-center">Rechercher</h1>
6      {{ form(formulaireRecherche) }}
7      <div class="columns is-multiline">
8        # Boucle de parcours des annonces = ici lister_annonces est la clé spécifiée dans le contrôleur #
9        # 'lister_annonces' => $annoncesRepository->findAll() #
10
11        {% for produit in resultatsRecherche %}
12          <div class="container shadow column is-4 mt-3">
13            <div class="card">
14              <div class="card-image">
15                <figure class="image is-4by3">
16                  
18                </figure>
19              </div>
20              <div class="card-content">
21                <div class="media">
22                  <div class="media-left">
23                    <figure class="image is-48x48">
24                      
25                    </figure>
26                  </div>
27                  <div class="media-content">
28                    <p class="title is-3has-text-info">Nom du vendeur
29                      : {{ produit.utilisateurs.email }}</p>
30                    <p class="title is-4">{{ produit.nomProduit }}</p>
31                    <p class="subtitle is-6">PRIX : {{ produit.prixProduit }} € </p>
32
33                    {% if produit.numero is not null %}
34                      <!-- alias + champ Produits (clé étrangère) + champ Table Reference (getter)--&gt;
35                      &lt;p class="has-text-info"&gt;Référence du produit (clé étrangère numero)
36                        : {{ produit.numero.numeroReference }}&lt;/p&gt;
37
38                    {% endif %}
39
40                    <!-- alias + champ Produits (clé étrangère) + champ Table Categories (getter)--&gt;
41                    &lt;p class="has-text-warning"&gt;Catégorie : (clé étrangère ManyToOne &amp; OneToMany)
42                      : {{ produit.categories.nomCategorie }}&lt;/p&gt;
43
44                    <!-- Plusieurs distributeur donc boucle for + champs table Distributeurs = alias + getter--&gt;
45                    {% for distributeur in produit.distributeurs %}
46                      &lt;p class="has-text-danger"&gt;Distributeur(s)
47                        :{{ distributeur.nomDistributeur }}&lt;/p&gt;
48                    {% endfor %}
49
50                    &lt;div class="content-footer mt-3 text-center"&gt;
51                      &lt;a href="{{ path('/produits/{id}', {'id': produit.id}) }}"
52                          class="button is-primary"&gt;Plus d'info&lt;/a&gt;
53                    &lt;/div&gt;
54
55                  &lt;/div&gt;
56                &lt;/div&gt;
57              &lt;/div&gt;
58            &lt;/div&gt;
59        {% endfor %}
60      &lt;/div&gt;
61    &lt;/div&gt;
62  {% endblock %}
</pre>

```

Afficher le formulaire

```

<h1 class="text-info text-center">Rechercher</h1>
{{ form(formulaireRecherche) }}

```

La boucle des résultats

```

{% for produit in resultatsRecherche %}
    <div class="container shadow column is-4 mt-3">
        <div class="card">
            <div class="card-image">
                <figure class="image is-4by3">
                    
                </figure>
            </div>

```

Afficher les données :

```

<div class="media-content">
    <p class="title is-3 has-text-info">Nom du vendeur
        : {{ produit.utilisateurs.email }}</p>
    <p class="title is-4">{{ produit.nomProduit }}</p>
    <p class="subtitle is-6">PRIX : {{ produit.prixProduit }} € </p>

    {% if produit.numero is not null %}
        <!--alias + champ Produits (cle étrangère) + champ Table Reference (getter)-->
        <p class="has-text-info">Reference du produit (cle étrangère numero)
            : {{ produit.numero.numeroReference }}</p>
    {% endif %}

    <!--alias + champ Produits (cle étrangère) + champ Table Categories (getter)-->
    <p class="has-text-warning">Catégorie : (cle étrangère ManyToOne & OneToMany)
        : {{ produit.categories.nomCategorie }}</p>

    <!--Plusieur distributeur donc boucle for + champs table Distributeurs = alias + getter-->
    {% for distributeur in produit.distributeurs %}
        <p class="has-text-danger">Distributeur(s)
            :{{ distributeur.nomDistributeur }}</p>
    {% endfor %}
    <div class="content-footer mt-3 text-center">
        <a href="{{ path('/produits/{id}', {'id': produit.id}) }}"
            class="button is-primary">Plus d'info</a>
    </div>

```

Résultats : une tranche de prix entre 40 et 500 €

Rechercher

Prix minimum du produit

Prix maximum du produit

Rechercher



 Nom du vendeur :  
michael@gmail.com

Iphone  
PRIX : 458,25 €



 Nom du vendeur :  
michael@gmail.com

Lego  
PRIX : 50,25 €

## Z-18 – LES ASSERTS ENTITE POUR SECURISE VOS TABLES

<https://symfony.com/doc/current/validation.html>

La validation est une tâche très courante dans les applications Web. Les données saisies dans les formulaires doivent être validées. Les données doivent également être validées avant d'être écrites dans une base de données ou transmises à un service Web.

Installer les dépendances :

```
composer require symfony/validator doctrine/annotations
```

Le validator est conçu pour valider des objets par rapport à *des contraintes* (c'est-à-dire des règles). Pour valider un objet, mappez simplement une ou plusieurs contraintes à sa classe, puis transmettez-la au validatorservice.

Dans les coulisses, une contrainte est simplement un objet PHP qui fait une déclaration assertive. Dans la vraie vie, une contrainte pourrait être : 'The cake must not be burned'. Dans Symfony, les contraintes sont similaires : ce sont des assertions qu'une condition est vraie. Étant donné une valeur, une contrainte vous indiquera si cette valeur respecte les règles de la contrainte.

Dans l'entité src/Entity/Produits.php

Ajouter les asserts dans les use :

use Symfony\Component\Validator\Constraints as Assert ;

```
/**
 * @ORM\Column(type="string", length=255)
 * @Assert\Length(
 *     min=6,
 *     max=50,
 *     minMessage="Le nom de l'annonce doit contenir au moins {{ limit }} caractères",
 *     maxMessage="Le nom de l'annonce ne doit pas dépasser {{ limit }} caractères"
 * )
 */
private $nom_produit;

/**
 * @ORM\Column(type="text")
 * @Assert\Length(
 *     min=10,
 *     max=1000,
 *     minMessage="La description de l'annonce doit contenir au moins {{ limit }} caractères",
 *     maxMessage="La description de l'annonce ne doit pas dépasser {{ limit }} caractères"
 * )
 */
private $description_produit;

/**
 * @ORM\Column(type="string", length=255)
 * @Assert\File(maxSize="6000000", maxSizeMessage="Le fichier est trop lourd ({{ size }} {{ suffix }}).
 * La taille maximale autorisée est : {{ limit }} {{ suffix }}"),
 */
private $image_produit;

/**
 * @ORM\Column(type="boolean")
 */
private $stock_produit;

/**
 * @ORM\Column(type="datetime")
 */
private $date_depot_produit;

/**
 * @ORM\Column(type="float")
 * @Assert\Positive
 */
private $prix_produit;
```

- Le nom du produit doit posséder au minimum 6 caractères et 50 maximum
- La taille de l'image est de 6000000 octet maximum soit 6 mb ou moins
- Etc...

La vue : comme prévu les asserts affiche bien les erreurs

## Ajouter un produit

Nom produit

a

Le nom du produit doit contenir au moins 10 caractères

Description produit

test

La description de l'annonce doit contenir au moins 10 caractères

Prix produit

-45

Cette valeur doit être strictement positive.

Image de l'annonce

BRAVO VOUS ETES DESORMAIS UN DEVELOPPEUR SYMFONY

LES +

- WEBPACK : <https://symfony.com/doc/current/frontend.html>
- API PLATFORM : <https://symfony.com/doc/current/the-fast-track/en/26-api.html>
- SYMFONY UX : <https://symfony.com/doc/current/frontend/ux.html>