# Mode opératoire : Démarrer avec Symfony

J’ai récemment réalisé une formation en ligne sur le Framework Symfony, autour des langages PHP et Twig. Excellente et très complète créée par Lior Chamla.

Formation de grande qualité que je recommanderai à qui souhaite découvrir Symfony.

[Symfony 5 : Le guide complet (web-develop.me)](https://learn.web-develop.me/view/courses/symfony-5-le-guide-complet-debutants-et-intermediaires)

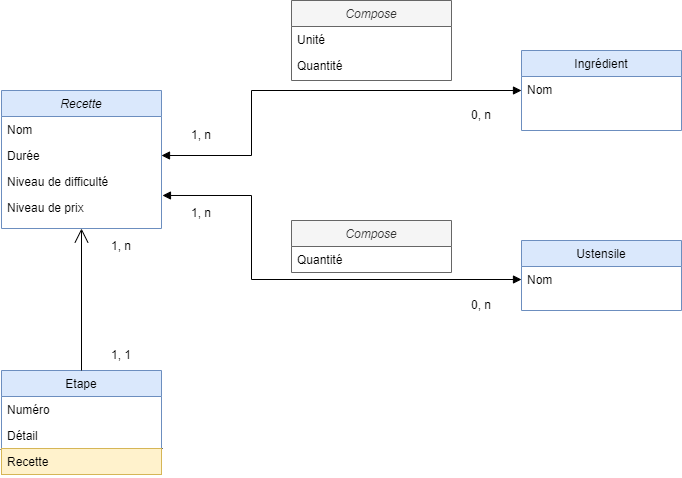
Afin de consolider et d’ancrer les compétences acquises par cette formation, je souhaite réaliser un projet et un document de synthèse permettant de faire le tour de toutes les pratiques et tous les enseignements de la formation.

L’idée est également de donner les éléments de base et les étapes à la construction d’un projet, les outils et les dépendances nécessaires à chaque étape, de créer un mode opératoire me permettant de démarrer rapidement sur n’importe quel projet.

Voici ma recette d’un développement avec Symfony étape par étape.

Je souhaite créer un projet de gestion de recettes de cuisine avec une architecture permettant de gérer les recettes de cuisines, les ingrédients et ustensiles nécessaires à la réalisation de la recette, ainsi que les différentes étapes de fabrication.

Le projet devrait permettre de consulter une liste de recette, créer ou modifier une recette, administrer la liste des ingrédients et ustensiles nécessaire à la réalisation d’une recette.



Ce projet pourrait à terme également évoluer en ajoutant le chargement de photos de la recette, un système de notation de recette ou la possibilité de laisser des commentaires.

## Étape 1 - Les outils nécessaires à la réalisation du projet

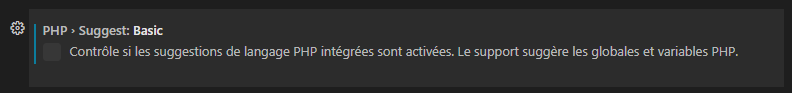
**Éditeur de texte VS Code**

**Extensions à installer :**

* Intelephense
* Namespace Resolver
* Twig Langage2
* MySQL

**Paramétrage de VS Code**

* Désactiver PHP>Suggest:Basic

****

* Activer le formatage lors de la sauvegarde

**Une image contenant texte

Description générée automatiquement**

* Définir le langage Twig

**Serveur Apache**

**PHP** (version minimale PHP7)

Connaitre sa version en ligne de commande : php -v

**Base de données MySQL**

**Composer** (outil de gestionnaires des dépendances)

Connaitre sa version en ligne de commande : composer -V

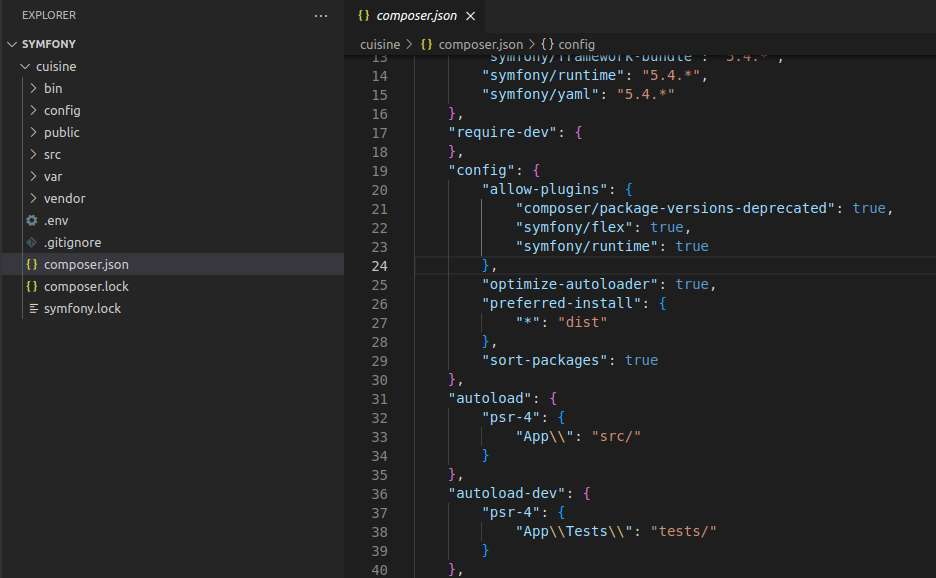
**Git** (outil de versionnage du code source)

Connaitre sa version en ligne de commande : git --version

## Étape 2 - Initialisation du projet

**Initialisation du projet**





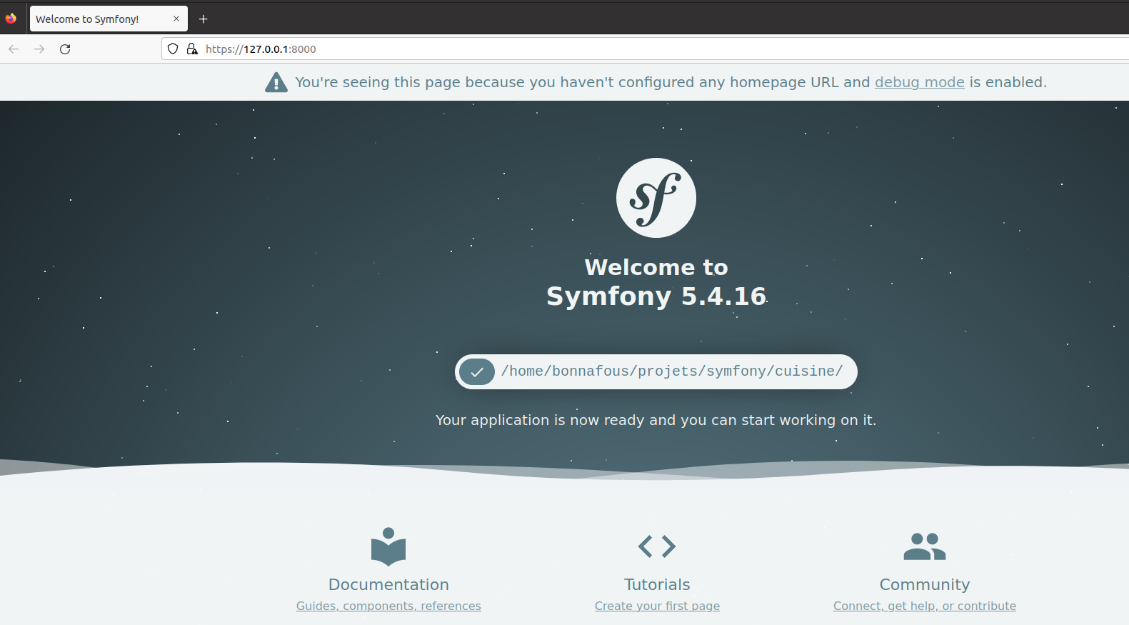
**Création d’un dépôt Git**



**Lancer le serveur**

**Une image contenant texte

Description générée automatiquement**

****

## Étape 3 - Les routes

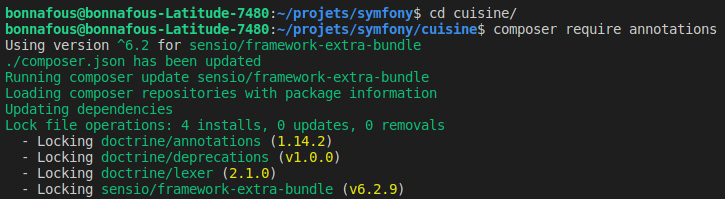
**Utilisation de Symfony Flex pour charger les services d’annotations**

[Using Symfony Flex to Manage Symfony Applications (Symfony 3.4 Docs)](https://symfony.com/doc/3.4/setup/flex.html)

Symfony Flex est une fonction permettant d’installer et gérer les applications Symfony. Il permet d’automatiser les tâches les plus communes et notamment la commande de chargement des services en récupérant les services correspondant à partir de mots clés (mailer, annotations, …) sans avoir à connaître le véritable nom des services.

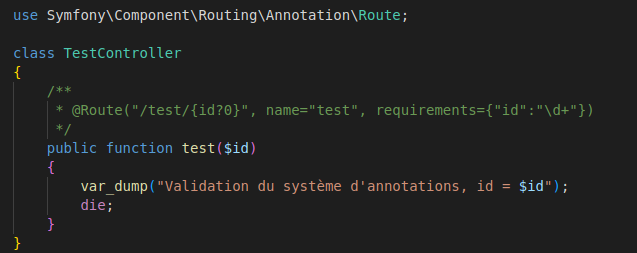
Par exemple lorsque je demande à composer de charger les services pour la gestion des annotations, Symfony Flex va installer automatiquement les bons services par défaut (doctrine/annotations, doctrine/deprecations, doctrine/lexer et sensio/framework-extra-bundle).

**Attention** à bien être positionné dans le répertoire du projet (sinon la commande de chargement ne fonctionne pas)

****

**Test du système d’annotations**

Nécessite l’utilisation de la bibliothèque de Route du service d’annotations de Symfony

****

**A compléter avec quelques routes du projet …**

## Étape 4 - Les objets de la base de données

**Création de la base de données**

**Installation du système de gestion des objets de base de données Symfony**

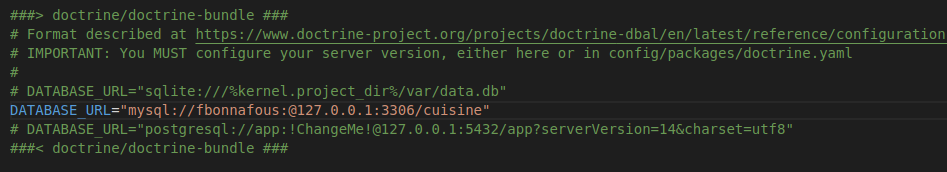
**Une image contenant texte

Description générée automatiquement**

**Une image contenant texte

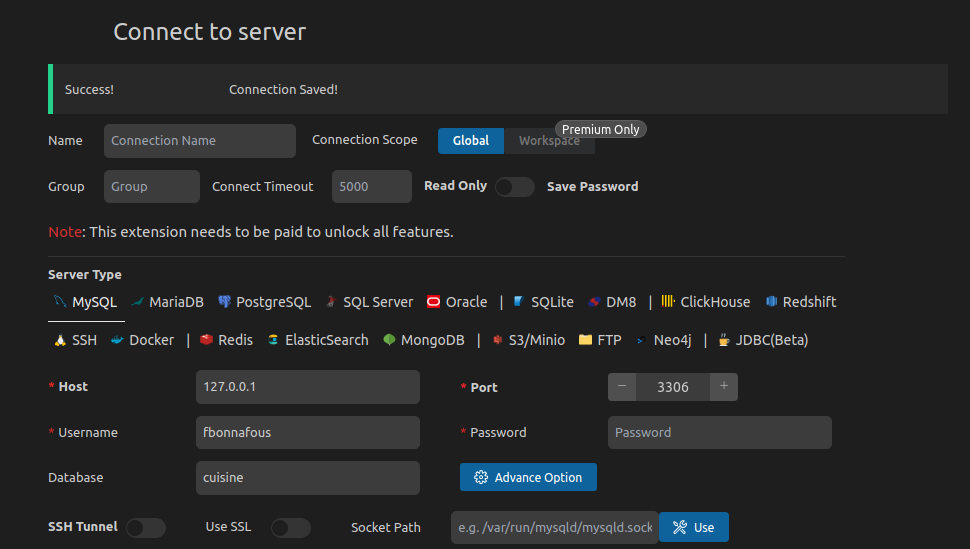
Description générée automatiquement**

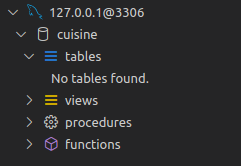
Symfony propose de mettre à jour le fichier d’environnement dans lequel je vais configurer ma base de données. Dans mon cas j’utilise une base de données MySQL pour l’environnement de développement.

****

**Création de la base de données pour le projet**

**Connexion à la base de données via l’extension MySQL de VS Code**

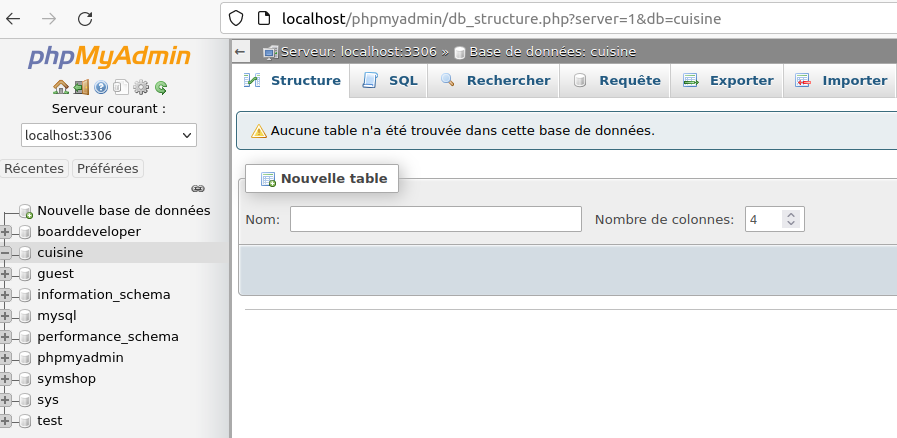
****

****

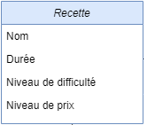
**Accès à la base de données via PHPMyAdmin**

PHPMyAdmin propose des options plus avancées que l’extension MySQL de VS Code. Il peut être intéressant d’y avoir accès.

Personnellement j’utilise l’extension MySQL de VS Code pour la visualisation de données (SELECT) et leurs manipulations de base (INSERT, UPDATE et DELETE). Et PHPMyAdmin pour des fonctionnalités plus avancées, telles que l’administration, la gestion d’import ou d’export.

****

**Création d’un objet simple**

****

**Installation des composant permettant la gestion des objets (maker)**

Maker va me permettre de créer mes entités en ligne de commande

**Une image contenant texte

Description générée automatiquement**

**Liste des commandes disponibles en lignes de commandes via maker**

**Une image contenant texte

Description générée automatiquement**

**Création de l’entité recette et de ses attributs**

**Une image contenant texte

Description générée automatiquementUne image contenant texte

Description générée automatiquement**

**Symfony crée une entité Recette.php ainsi qu’un Repository**

**Une image contenant texte

Description générée automatiquement**

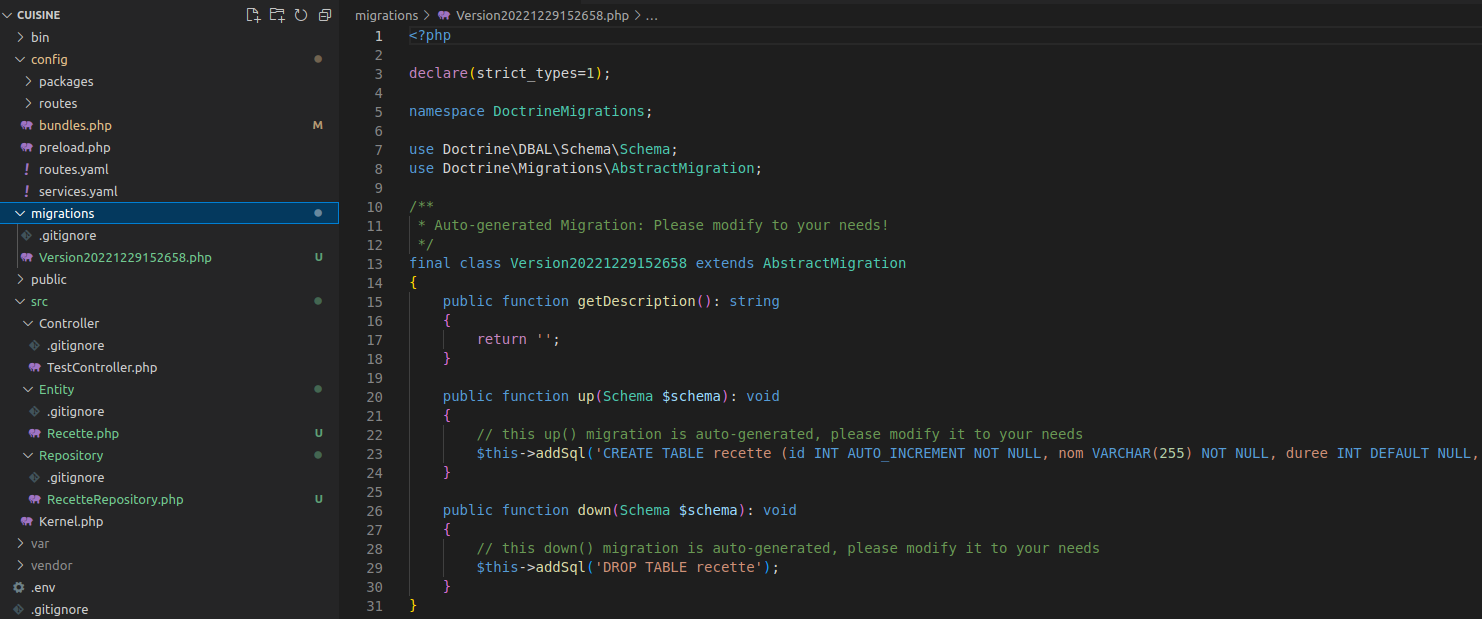
**Une image contenant texte, capture d’écran, moniteur, équipement électronique

Description générée automatiquement**

Il faut ensuite réaliser une migration. La migration va générer un script contenant les instructions SQL nécessaires à la création de l’objet dans la base de données.

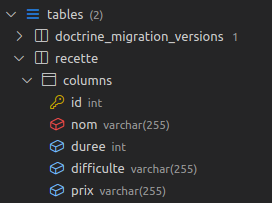
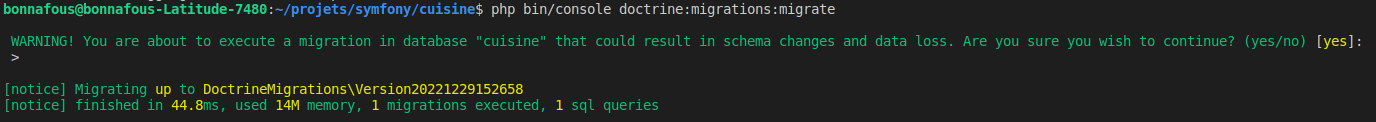
**Une image contenant texte

Description générée automatiquement**

****

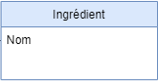
Une fois le script de migration en base, il reste à l’exécuter.

L’exécution du script de migration va créer l’entité dans la base de données.

****

Il reste à créer les objets ingrédients et ustensiles qui vont stocker les éléments nécessaires à nos recettes de cuisine.

Créer les entités, créer les scripts de migration et exécuter les scripts de migration pour mettre à jour notre base de données.



Une image contenant texte

Description générée automatiquement

**Création d’un objet avec un relation de type 1-n**

**Création d’un objet avec un relation de type n-n**

## Étape 5 - Alimenter la base de données (les fixtures)

Les fixtures sont des jeux de données. Symfony permet d’alimenter la base de données via l’utilisation de jeux de données.

L’installation du gestionnaire de fixtures va créer un répertoire de gestion de fixture dans lequel il va être possible de développer un script (AppFixtures.php) permettant de mettre à jour la base de données.

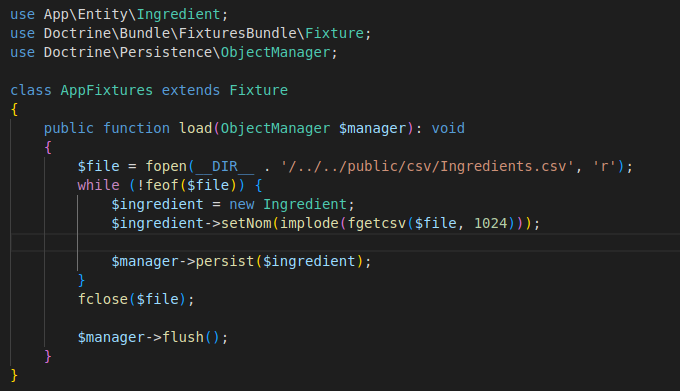
**Installation du système de gestion des fixtures**

**Une image contenant texte

Description générée automatiquement**

****

Afin d’alimenter la base de données j’ai créé deux jeux de données au format CSV qui contiennent une liste d’ingrédients et une liste d’ustensiles que je vais vouloir importer dans la base de données.

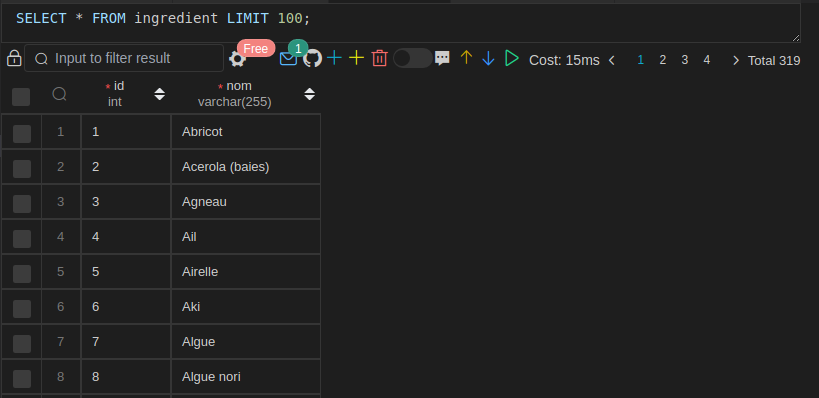
****

Une fois le script crééil suffit de l’exécuter en ligne de commande.

**Une image contenant texte

Description générée automatiquement**

L’exécution du script permet d’alimenter les tables de la base de données.

****

## Étape 6 - Le langage d’affichage (Twig)

Twig est un moteur de template pour PHP utilisé avec Symfony.

**Installation du moteur de template Twig**

Une image contenant texte

Description générée automatiquement

L’installation met à jour les fichiers de configuration et crée les répertoires dédiés aux templates.



Création de la page d’accueil

Une image contenant texte, équipement électronique, capture d’écran

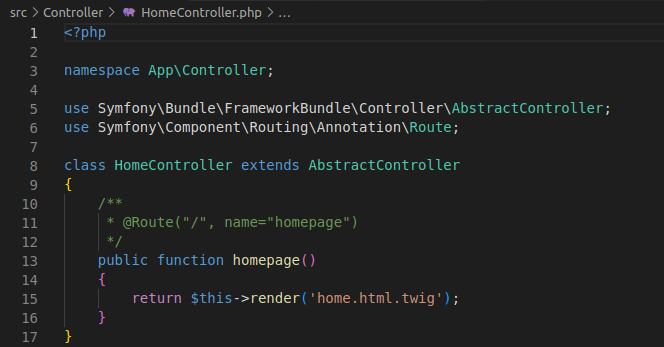
Description générée automatiquement

**Création du contrôleur de la page d’accueil**

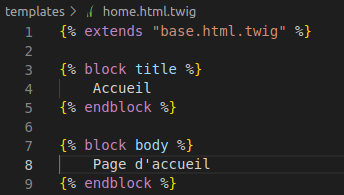
La classe HomeController hérite d’une classe AbstractController qui permet d’hériter de fonctionnalités avancées plus facile à écrire.

Dans cet exemple l’ AbstractController permet d’utiliser une fonction render qui redirige sur la page home.html.twig.

[symfony/AbstractController.php at 6.2 · symfony/symfony · GitHub](https://github.com/symfony/symfony/blob/6.2/src/Symfony/Bundle/FrameworkBundle/Controller/AbstractController.php)

****

**Création du template**



## Étape 7 - Premiers formulaires (mise en place de formulaires simples avec CRUD)

CRUD est l’acronyme de Create, Read, Update, Delete.

Symfony dispose d’une commande permettant la gestion du CRUD.

La commande de génération du CRUD permet de créer :

* le contrôleur pour l'entité avec des fonctions permettant
  + la visualisation de la liste des ingrédients
  + la visualisation des informations d’un ingrédient
  + la création d'un nouvel ingrédient
  + la mise à jour d’un ingrédient
  + la suppression d’un ingrédient
* un formulaire par défaut
* des templates permettant de visualiser, créer ou modifier les informations

**Installation du composant de sécurité de validation des formulaires csrf**

**Une image contenant texte

Description générée automatiquement**

**Création des composant à partir de la commande**

**Une image contenant texte

Description générée automatiquement**

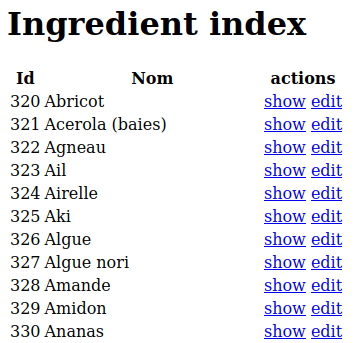
****

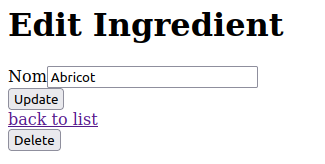
**Contenu du contrôleur**

**Une image contenant texte

Description générée automatiquement**

**Exemples de rendu (liste et formulaire de modification d’un ingrédient)**

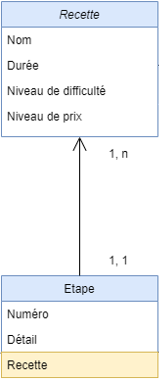
****

****

## Étape 8 - Les objets de la base de données de type 1-n

Un objet de type 1-n est un objet qui dispose d’une relation avec un autre objet.

* Une recette de cuisine peut être composé de plusieurs étapes pour sa réalisation (n)
* Une étape est propre à une seule recette de cuisine (1)
* Dans ce genre de relation l’identifiant de la recette devient un élément clé (clé secondaire) de l’entité étape.



Création de l’entité Étape

Une image contenant texte

Description générée automatiquement

Création de la relation avec l’entité Recette

Une image contenant texte

Description générée automatiquement

Symfony crée de nouvelles fonctions au niveau de l’entité recette pour permettre de faciliter la gestion des étapes.

Une image contenant texte

Description générée automatiquement

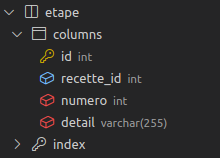
Reste à créer le script de migration et à l’exécuter pour retrouver la nouvelle entité et la relation au niveau de la base de données.

Une image contenant texte

Description générée automatiquement



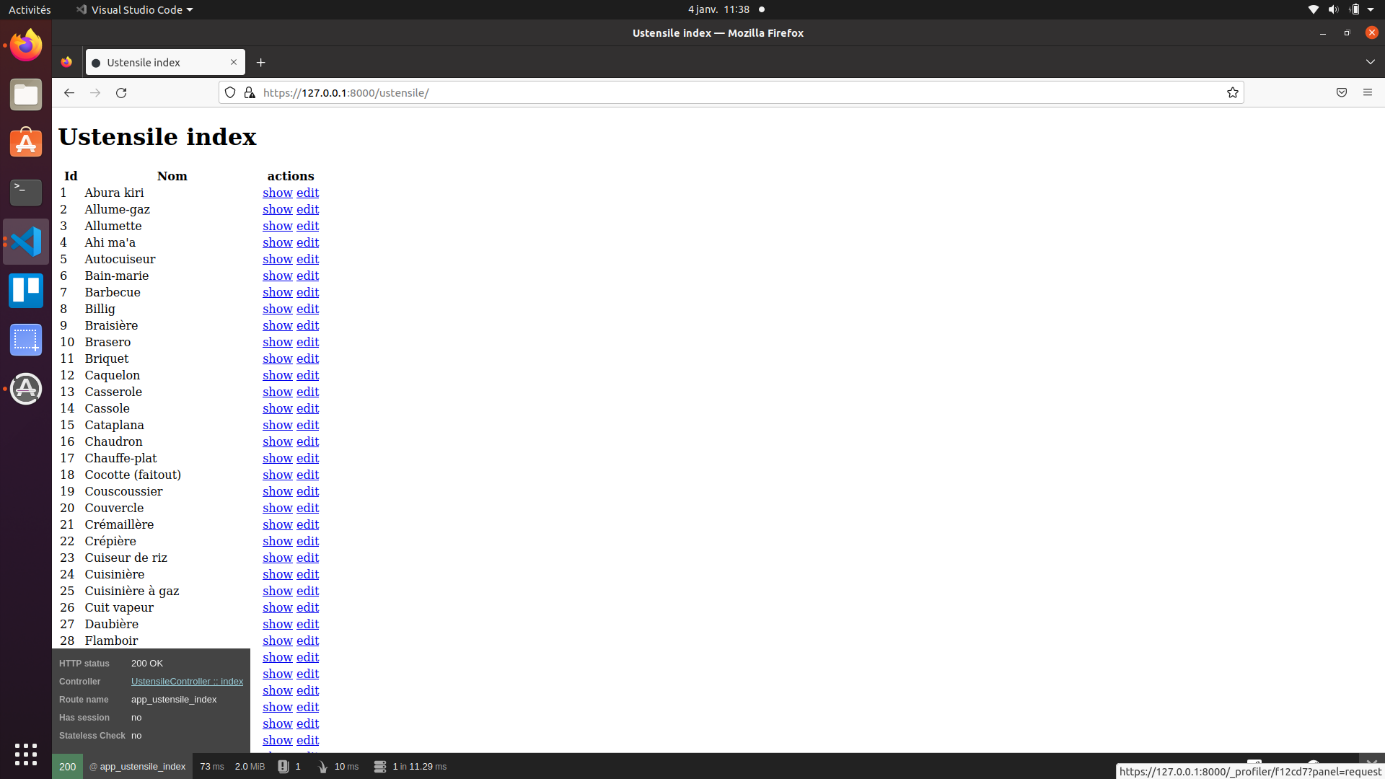
Au niveau de la base de données



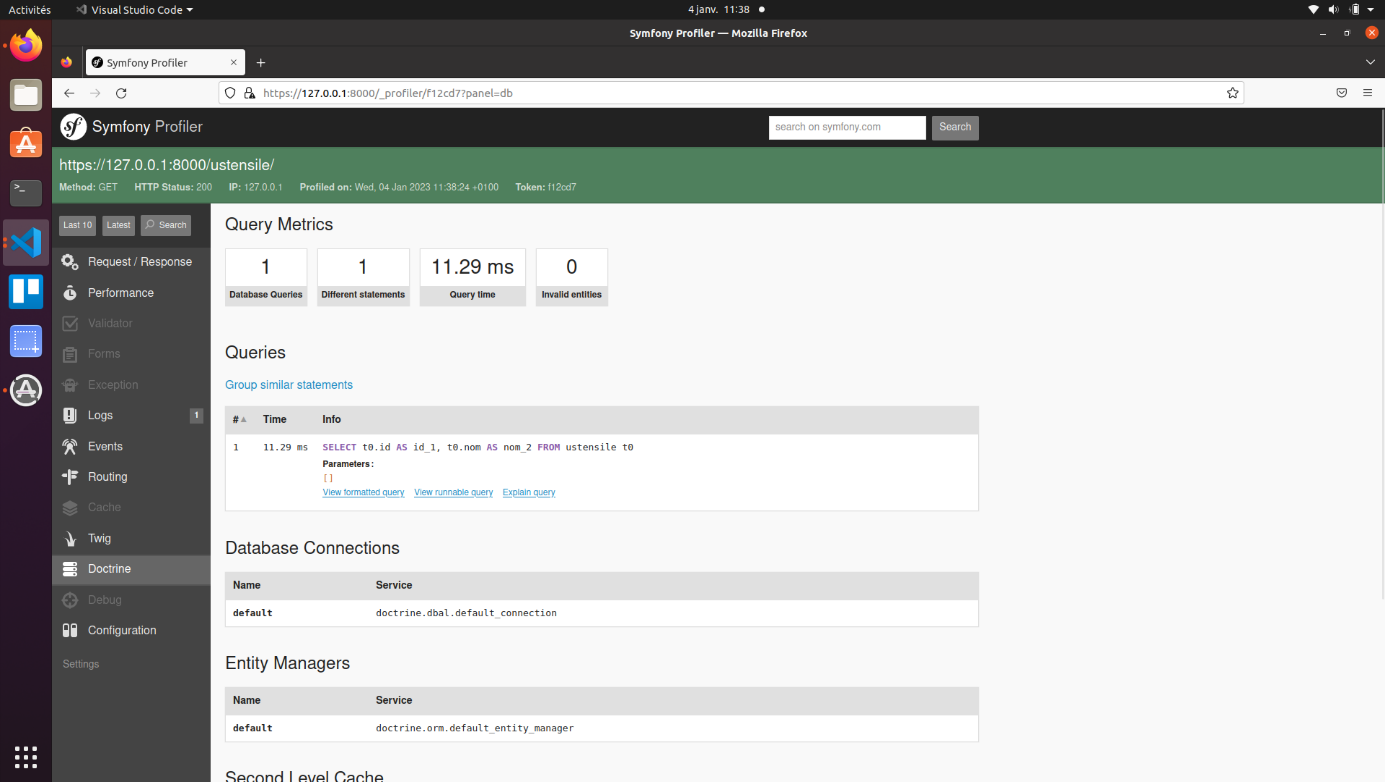
## Bonus 1 - Le debugger de Symfony

Avant de continuer le développement de l’application, on peut installer un composant sympa de Symfony qui communique de nombreuses informations sur le traitement des requêtes, les performances, les accès à la base de données et plein d’autres informations, le debugger.









## Étape 9 - Mise en place d’un service d’authentification

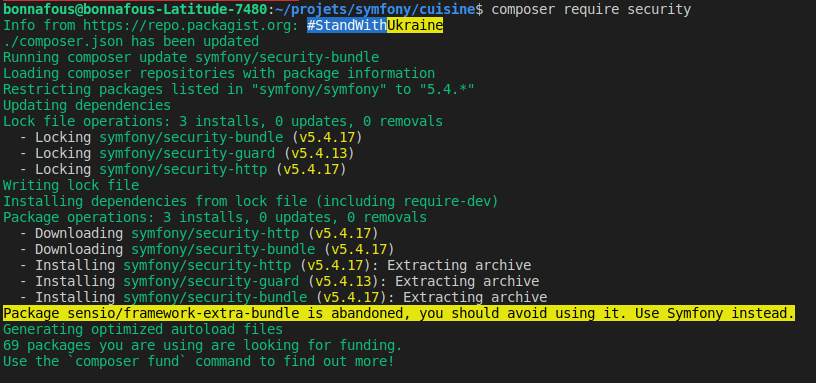
Afin de savoir quelle partie mon utilisateur à le droit de consulter et quelles actions il a le droit de réaliser, nous allons créer un système d’authentification avec les rôles administrateur et le rôle utilisateur connecté et un rôle de visiteur.

Le visiteur pourrait visualiser les recettes.

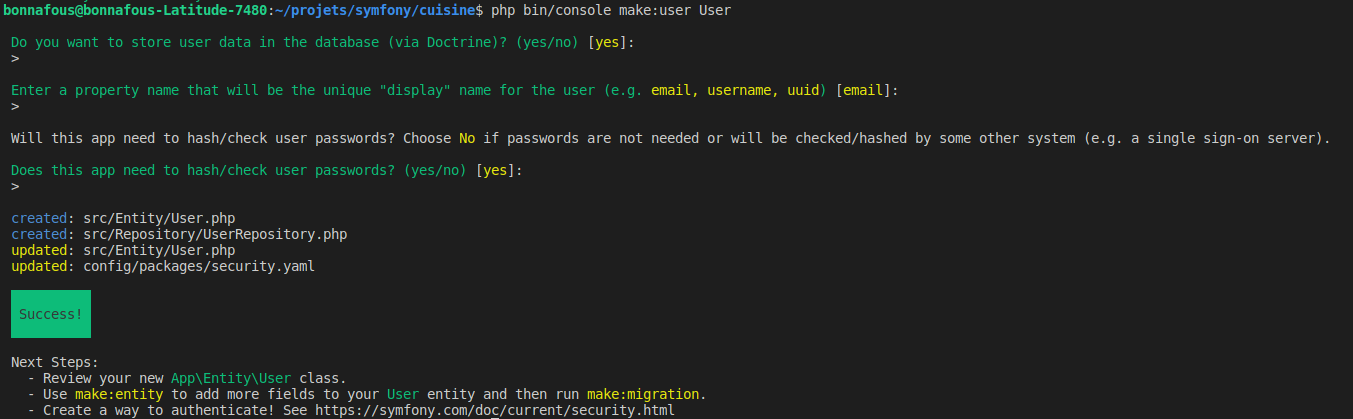
L’utilisateur pourrait créer des recettes et les visualiser.

L’administrateur pourrait administrer les ingrédients et ustensiles, créer des recettes et les visualiser.

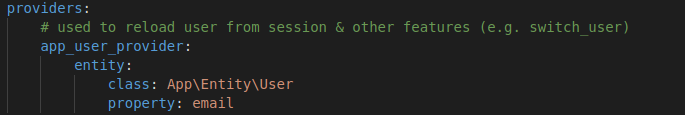
Il faut d’abord installer le service sécurité de Symfony.



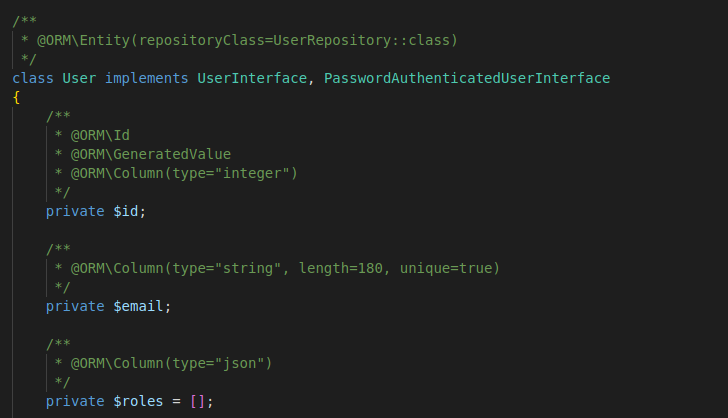
Une fois installé nous allons créer une entité pour gérer nos utilisateurs. Il s’agit d’une entité un peu particulière créée à partir de la commande make:user (au lieu de la commande make :entity) qui implémente l’interface UserInterface.



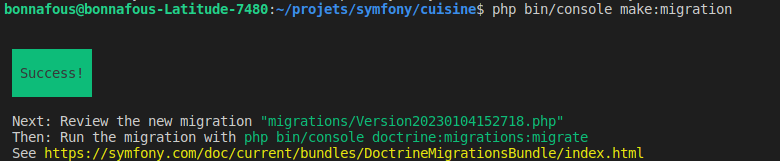
La commande va modifier le fichier de sécurité yaml pour indiquer quelle classe va gérer les utilisateurs et quelle propriété va permettre leur authentification.



La commande va créer une classe permettant de gérer nos utilisateurs.



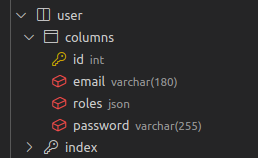
On effectue la migration qui va créer le fichier de migration contenant les commande SQL permettant la mise à jour de notre base de données avec la création de notre entité stockant les utilisateurs.



Et on lance le traitement.



Notre entité utilisateur est désormais créée au niveau de la base de données

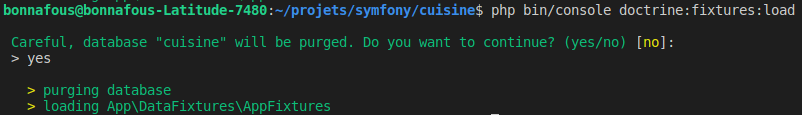


Reste à créer un jeu de données d’utilisateurs

*Une image contenant texte

Description générée automatiquement*

On lance la création du jeu de données.



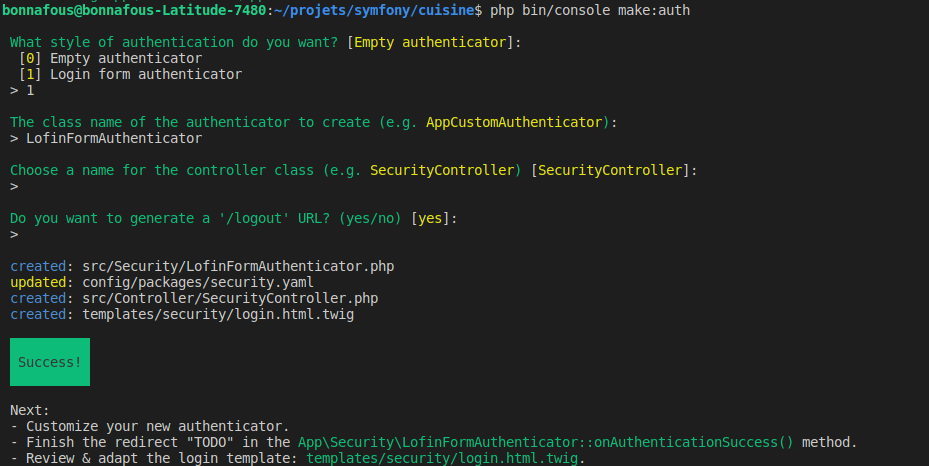
Nos deux utilisateurs (utilisateur connecté et administrateur) existent maintenant dans notre base de données.

Une image contenant texte, moniteur, écran, noir

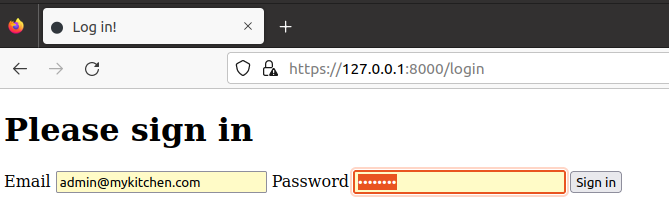
Description générée automatiquement

Reste à créer le système d’authentification.

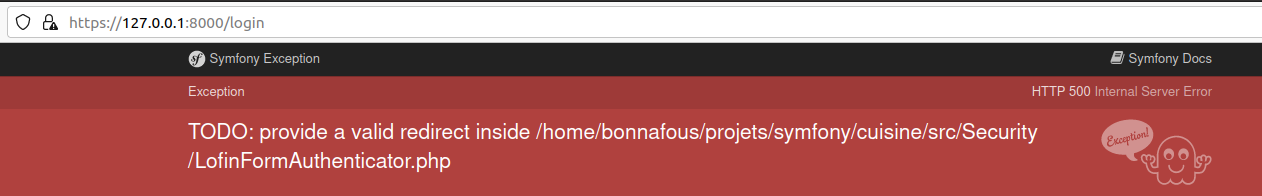
*A corriger j’ai tapé lofin au lieu de login (c’est moche)*



L’utilisateur peut se connecter grâce au formulaire d’authentification, à l’aide de son courriel



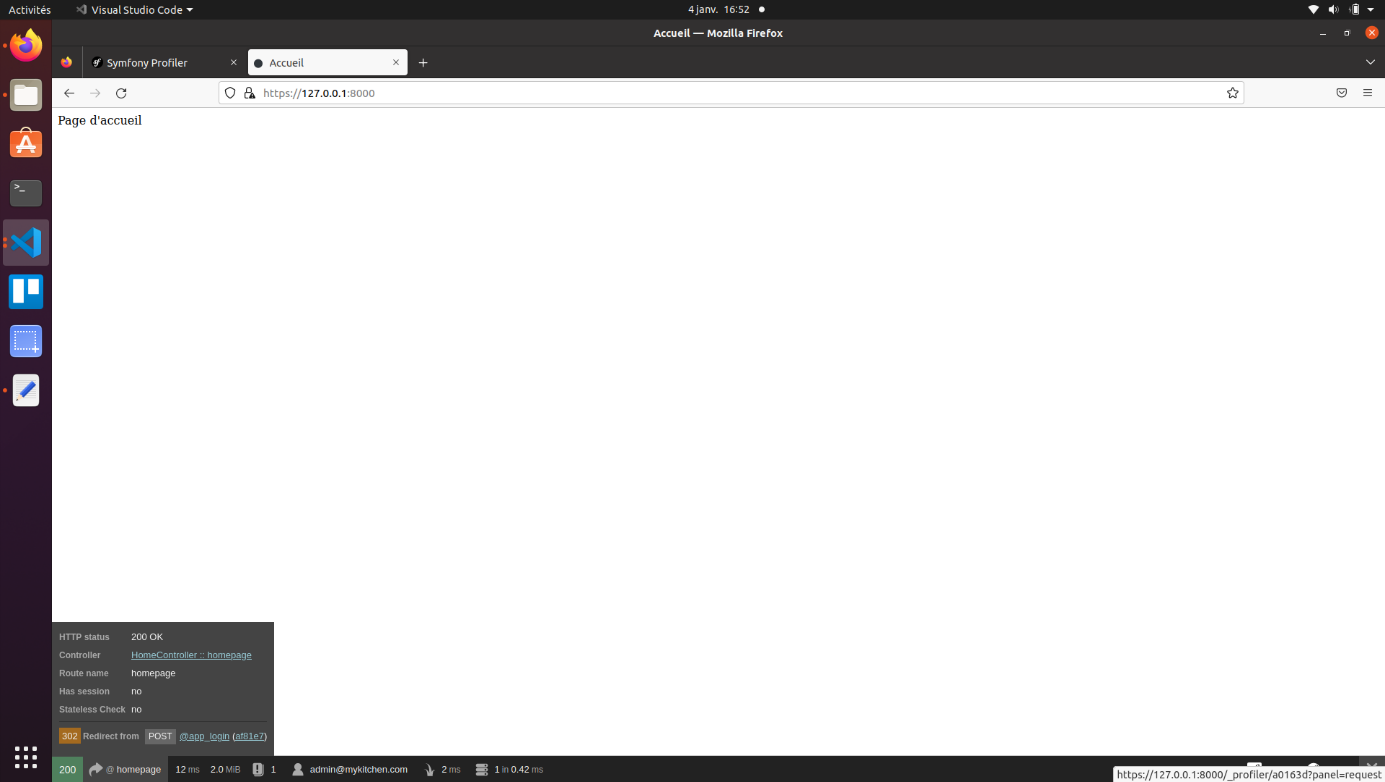
Dernier détail, il reste à rediriger l’utilisateur une fois connecté.



Une image contenant texte, moniteur, capture d’écran, écran

Description générée automatiquement

Cette fois l’administrateur est bien connecté et nous l’avons redirigé vers la page d’accueil.



## Étape 10 - La gestion des rôles

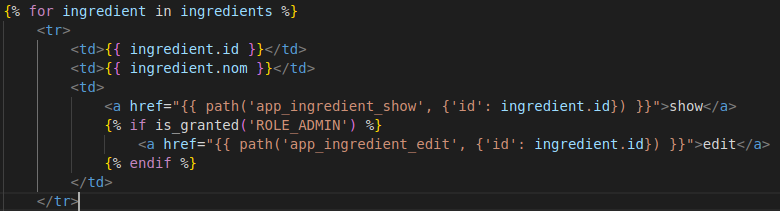
Symfony offre la possibilité de permettre l’accès aux fonctions, à une partie du site ou à un rendu d’une page en fonction des rôles défini.

Dans le cas de mon application, je gère trois rôles : administrateur, utilisateur connecté et visiteur.

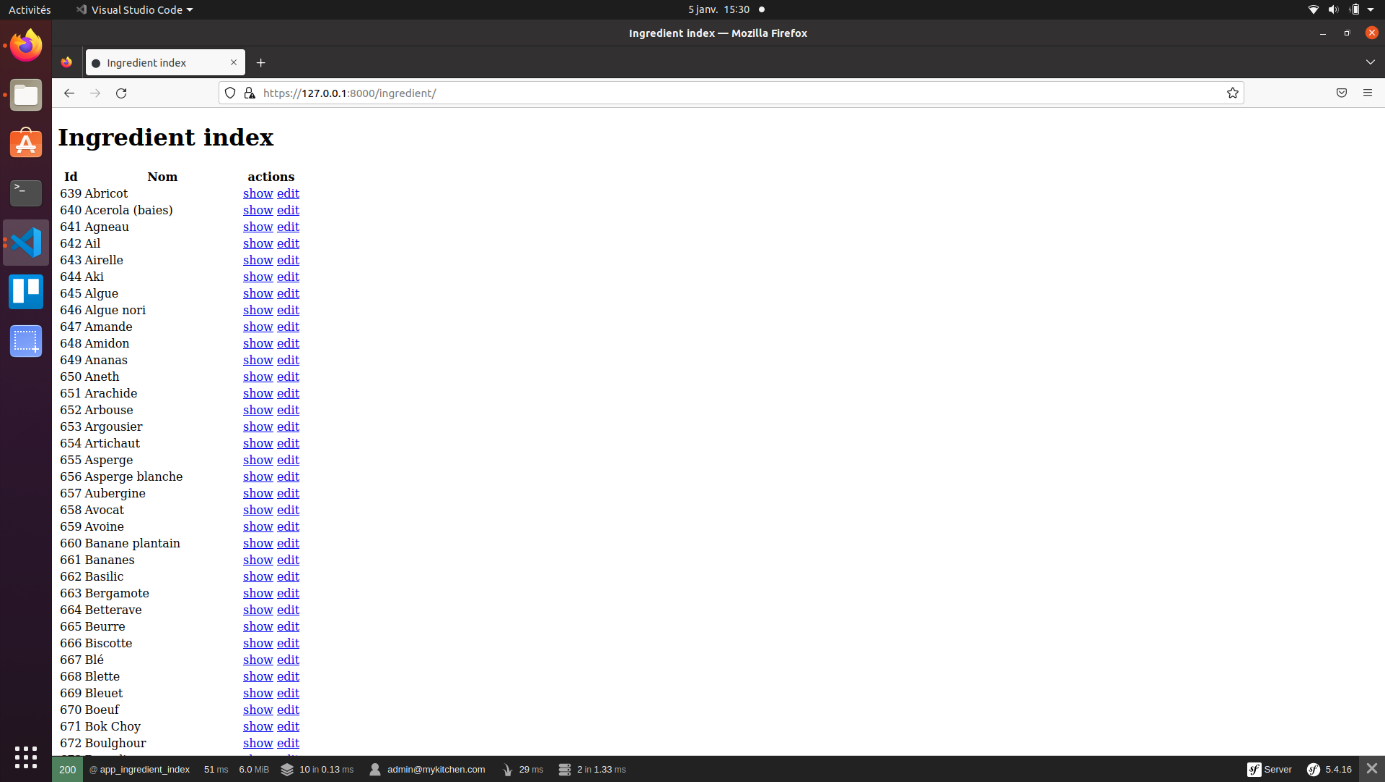
**Gestion des rôles au niveau de Twig**

Les rôles peuvent être gérés au niveau du rendu de la page grâce aux fonctions de Twig.

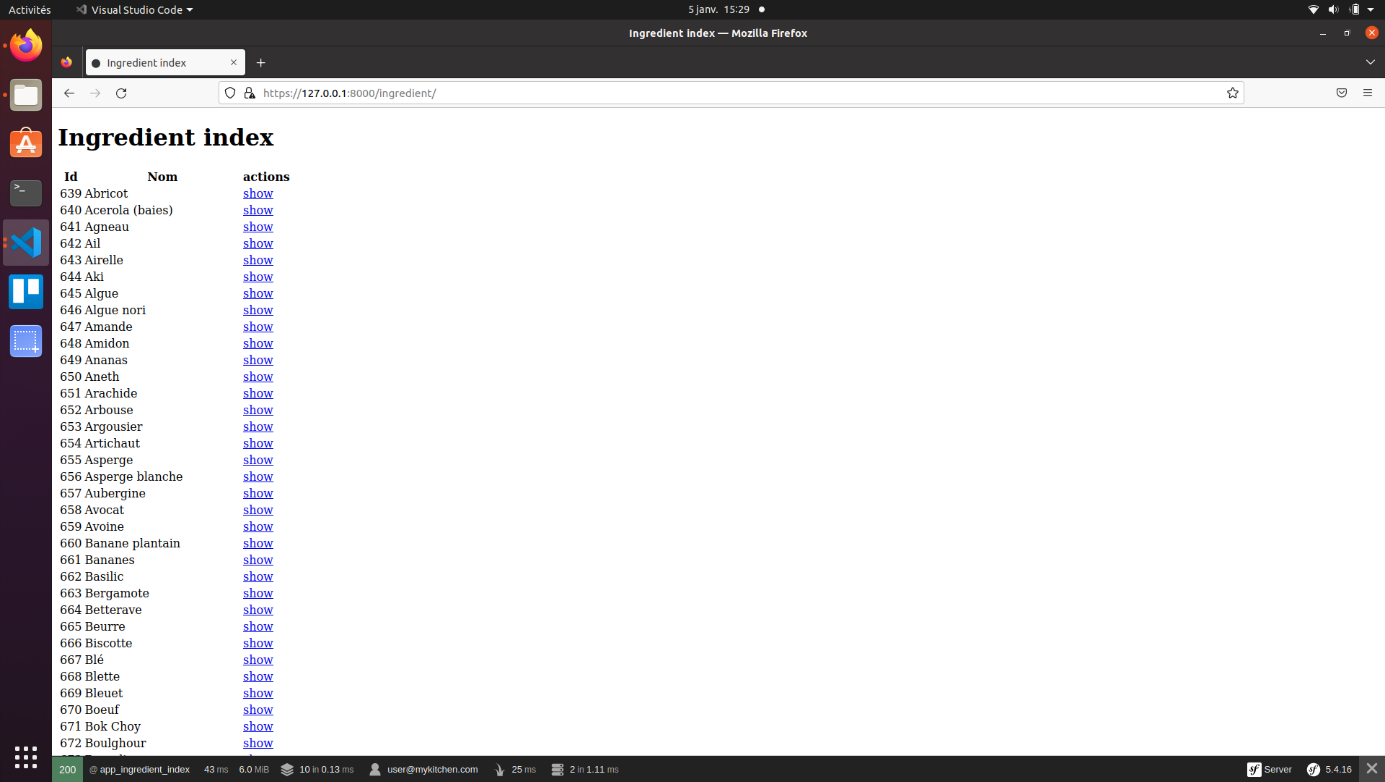
On peut masquer la fonction d’édition d’un ingrédient si l’on n’a pas les droits d’administration en conditionnant l’accès à l’administrateur (is\_granted) ou à l’utilisateur connecté (app.user).



En mode administrateur



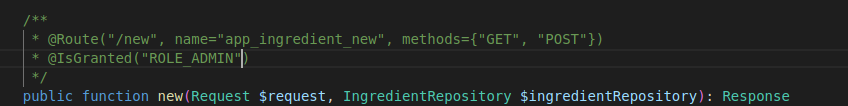
En mode utilisateur connecté ou visiteur



**Gestion des rôles au niveau du contrôleur**

On peut également conditionner restreindre les droits d’une fonction particulière.

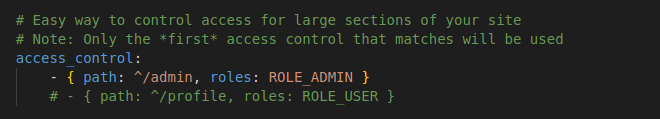
L’utilisateur qui tenterait d’accéder à la fonction pour définir un nouveau produit directement en modifiant l’URL ne pourrait y accéder que s’il a un rôle d’administrateur. Par défaut Symfony renvoi l’utilisateur à la page d’identification le cas échéant.



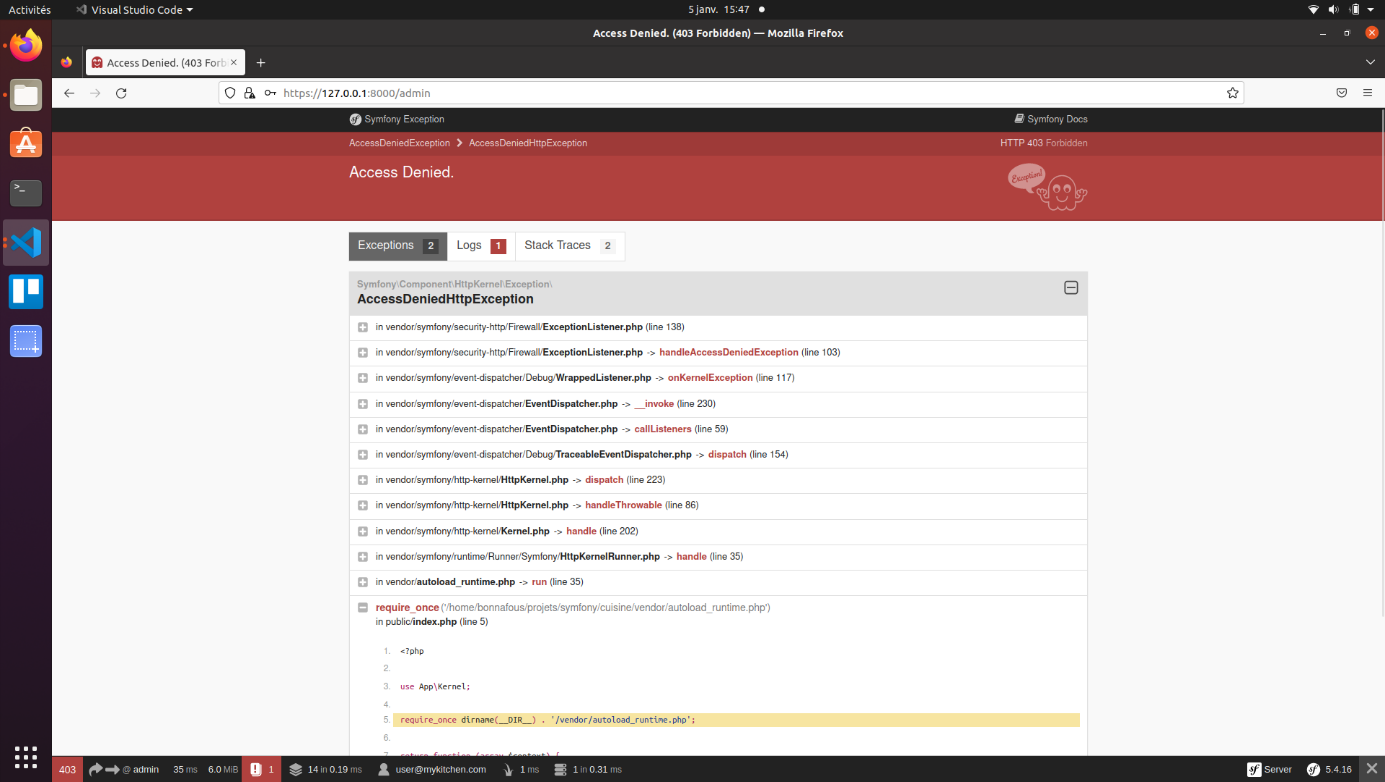
**Gestion des rôles au niveau de la configuration de la sécurité**

Il est également possible au niveau du fichier de configuration de réserver une partie du site à un rôle particulier. On pourrait envisager que les URL commençant par /admin soient dédiées à l’administration et accessible uniquement aux utilisateurs ayant le rôle d’administrateur.

La configuration se fait alors au niveau du fichier de sécurité (security.yaml).



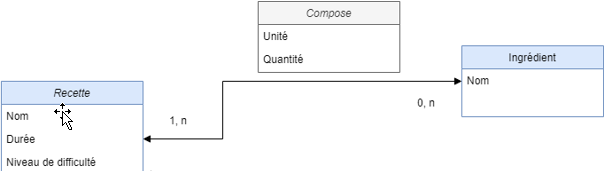
Par défaut un message d’erreur indique que l’accès n’est pas autorisé si un utilisateur autre qu’un administrateur essaye d’accéder à la ressource.



## Étape 11 - Les objets de la base de données de type n-n

Un objet de type n-n est un objet qui dispose d’une relation avec un autre objet.

* Une recette de cuisine peut être composé de plusieurs ingrédients pour sa réalisation (n)
* Un ingrédient peut composer plusieurs recettes de cuisine (n)
* Dans ce genre de relation, on crée une entité entre les entités ingrédient et recette qui va gérer la relation et permettre de stocker des attributs propres à la relation (par exemple la quantité d’un ingrédient pour la recette et son unité, ma recette de crêpe est composée de 300 grammes de sucre).



Symfony offre deux possibilités pour gérer ce type de relation :

* L’utilisation de la relation ManyToMany (ne permet pas de gérer les attributs)
* La création d’une nouvelle entité qui va permettre de gérer des attributs

Dans mon exemple, il est nécessaire de créer une nouvelle relation du fait que je souhaite gérer des attributs dans ma relation.

Création de l’entité gérant la relation entre les ingrédients et la recette

La simulation du ManyToMany se fait par la création de ManyToOne sur les deux identifiants qui seront la clé de la nouvelle entité.

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Création des attributs quantité et unité

Une image contenant texte

Description générée automatiquement

Génération de script de migration

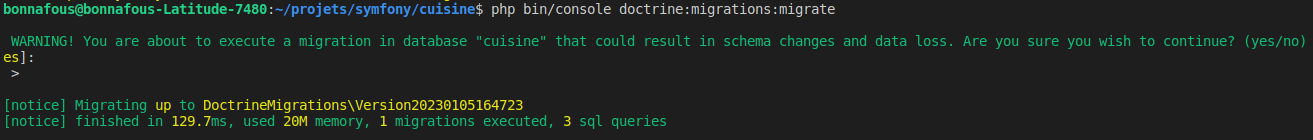
Une image contenant texte

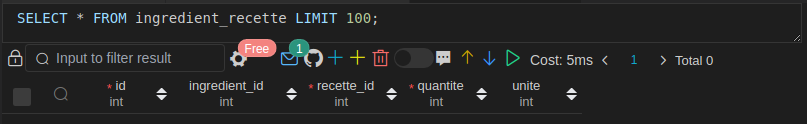
Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Mise à jour de la base de données





## Bonus 2 - Les messages Flash

L’abstract contrôleur permet la création et la gestion de messages flash, messages que nous pourrons utiliser pour informer l’utilisateur de certaines actions lors du traitement de formulaires.

Au niveau du contrôleur on a la possibilité d’ajouter un message par exemple lorsque le formulaire est soumis.

Une image contenant texte

Description générée automatiquement

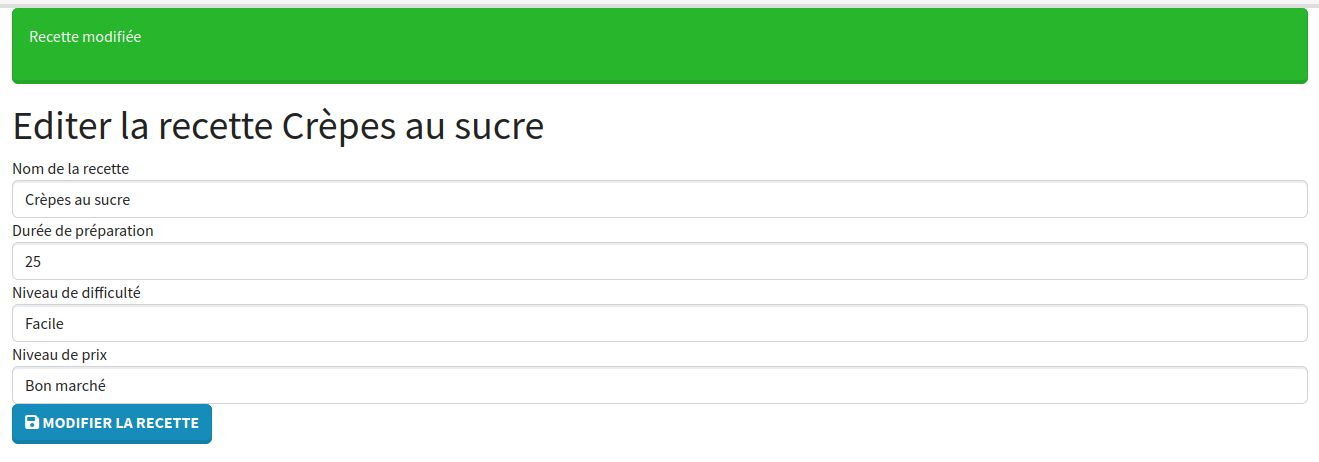
Au niveau de l’affichage les messages sont stockés dans une variable app.flashes. Les messages sont stockés en mémoire jusqu’à être affichés.

Modification du programme base.html.twig dont hérite tous mes fichiers Twig

Une image contenant texte

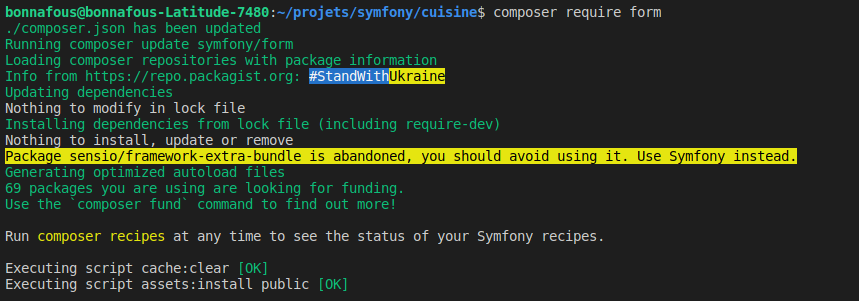
Description générée automatiquement

Résultat de l’affichage d’un message



## Étape 12 - Les formulaires simples

La gestion des formulaires nécessite la mise à jour du composant.



Au niveau du répertoire de gestion des formulaires, on définit les propriétés des champs du formulaire.

Une image contenant texte

Description générée automatiquement

Au niveau du contrôleur nous créons une fonction que permet :

* D’afficher le formulaire
* De traiter la soumission du formulaire

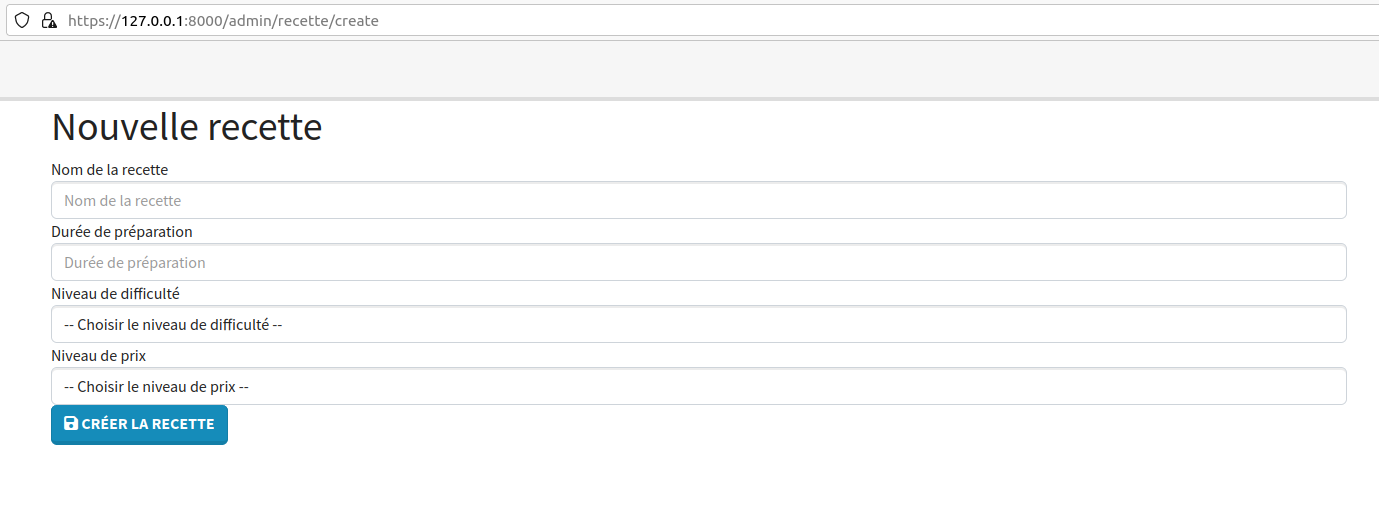
*Une image contenant texte

Description générée automatiquement*

Affichage au niveau de Twig

Une image contenant texte

Description générée automatiquement



## Bonus 3 - Amélioration du visuel avec Bootstrap

Définition d’un thème au niveau du fichier de configuration

Une image contenant texte

Description générée automatiquement

Chargement des feuilles de style dans le fichier de base de l’ensemble des pages du site

Une image contenant texte

Description générée automatiquement

## Étape 12 – Des formulaires un peu plus complexes (la relation OneToMany)

Chaque recette est généralement constituée de plusieurs étapes, aussi nous allons créer un formulaire avec la possibilité d’ajouter dynamiquement le nombre d’étapes voulues.

Pour cela, j’ai suivi un petit tutoriel de Symfony qui explique comment faire. Pour ajouter un peu de dynamisme à la création des étapes, il faudra ajouter au PHP, des scripts JS et faisant appel à JQuery.

[How to Embed Forms (Symfony Docs)](https://symfony.com/doc/current/form/embedded.html)

Création d’un formulaire pour l’affichage des étapes

Une image contenant texte

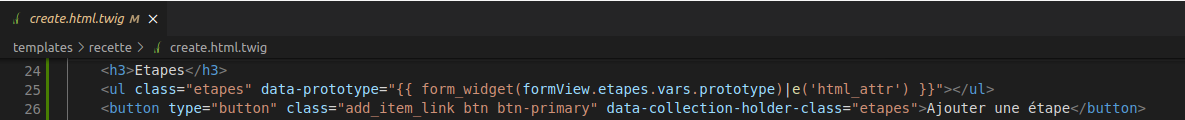
Description générée automatiquement

Modification du formulaire de recette pour intégrer les étapes

Une image contenant texte

Description générée automatiquement

Au niveau du formulaire de création Twig, ajout des étapes de la recette



Pour rendre le formulaire dynamique nous avons besoin d’utiliser la bibliothèque de fonctions JQuery que nous déclarons dans le formulaire de base, ainsi qu’un script JS que nous allons écrire permettant d’ajouter des éléments à une collection (les étapes de notre recette).

Une image contenant texte

Description générée automatiquement

Au niveau de notre entité, il faut ajouter une fonction \_\_tostring et dire que la relation avec l’entité étape persiste.

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Le script JS permet d’ajouter ou supprimer une étape à la recette

Une image contenant texte

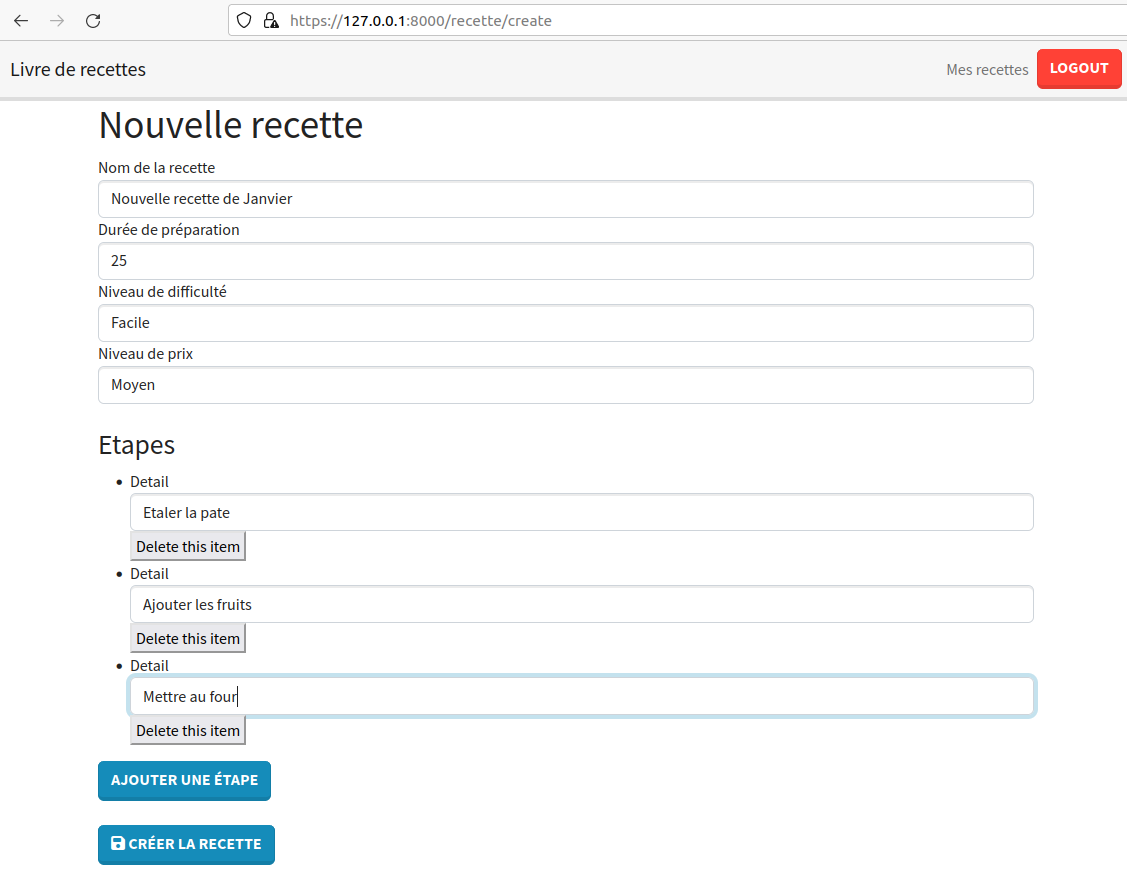
Description générée automatiquement

Au niveau du contrôleur on rajoute la relation entre les étapes et la recette, en rattachant les étapes au moment de la validation du formulaire.

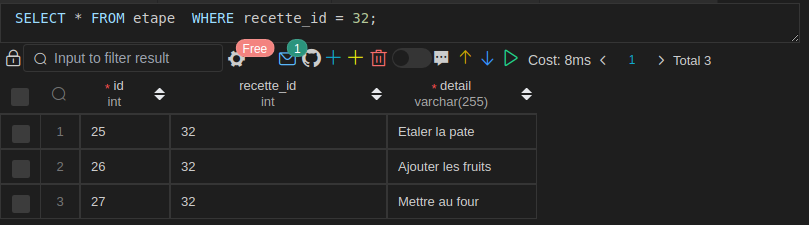
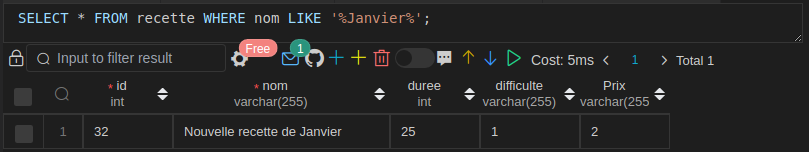
Une image contenant texte

Description générée automatiquement

A l’exécution le navigateur permet de saisir une recette, d’ajouter ou supprimer des étapes, et d’enregistrer le résultat dans la base de données.



Résultat du traitement au niveau de la base de données

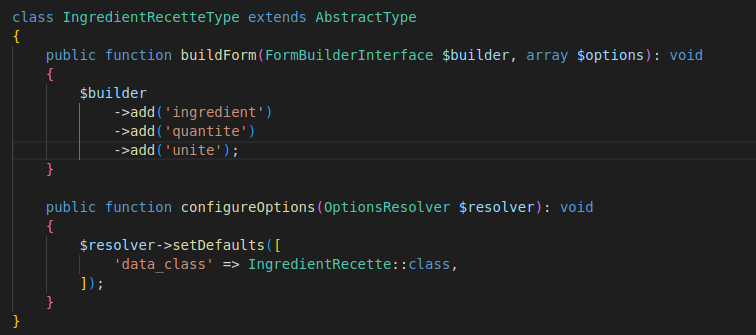


## Étape 13 - Des formulaires un peu plus complexes (la relation ManyToMany)

J’ai un peu ramé sur cette partie, ayant renommé en masse certains champs pour avoir quelque chose de plus propre, je suis tombé sur des erreurs de type « The mappings are inconsistents » qui m’ont obligé à revoir en profondeur mes classes Entités. Ce qui m’a aussi permis de mieux comprendre leur fonctionnement.

Chaque recette est composée d’ingrédients et chaque ingrédient peut composer plusieurs recettes. Ces ingrédients sont stockés dans une entité intermédiaire qui fait la relation entre Ingrédient et Recette.

Nous allons créer un formulaire qui contient les information de la relation intermédiaire : l’ingrédient qui compose la recette, sa quantité et son unité.



Nous ajouton cette collection à notre formulaire de recette.

Une image contenant texte

Description générée automatiquement

Au niveau du contrôleur, il reste à ajouter les ingrédients à la recette.

Une image contenant texte

Description générée automatiquement

Au niveau du visuel nous réutilisons le script JS pour prendre en compte l’ajout d’ingrédients à notre recette.

Une image contenant texte

Description générée automatiquement

Une image contenant texte, moniteur, écran, noir

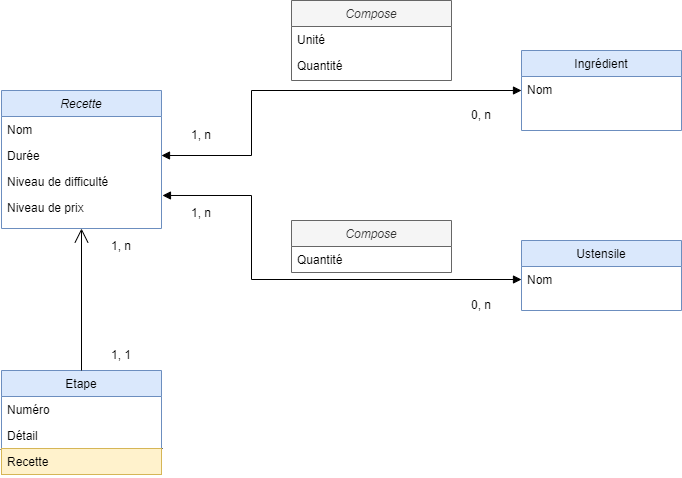
Description générée automatiquement

Une image contenant texte, moniteur, écran, noir

Description générée automatiquement

Une image contenant texte, moniteur, noir, écran

Description générée automatiquement



## Étape 14 - Améliorer notre petite application

Il s’agit d’une rapide introduction à Symfony sur un exemple assez simple de mise en place de composants.

De nombreux points sont à terminer (formulaires d’édition, de suppression) ou peuvent être améliorés aussi bien au niveau visuel, qu’au niveau des fonctionnalité.

On pourrait imaginer de gérer des sous catégories d’ingrédients, d’afficher une image pour chaque ingrédient ou ustensile.

D’améliorer la gestion des droits, la navigation, l’expérience utilisateur, le design, un système de notation des recettes, on peut imaginer plein de choses :D

L’idée de ce développement était surtout d’avoir un guide pour démarrer facilement un projet avec Symfony.

## Bonus 4 - Le composant de gestion de mail

La première chose à faire pour pouvoir générer des mails est d’installer les composants pour l’utiliser.

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

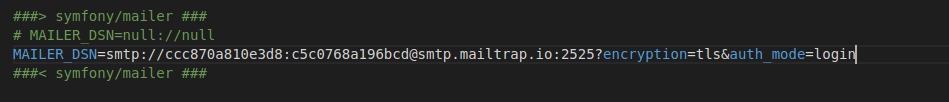
J’ai repris l’application mailtrap.io pour faire mes tests, comme proposé par Lior dans sa formation. Assez pratique car cela demande peu de configuration.

Juste mettre à jour le MAILER\_DNS au niveau du fichier d’environnement.

*Todo - A flouter*

Une image contenant texte

Description générée automatiquement



Au niveau du contrôleur de recette, charger la mailer interface au niveau du constructeur

Une image contenant texte

Description générée automatiquement

Définir un mail après la création d’une nouvelle recette, et l’envoyer.

## 

Définir un template pour le corps du mail

## 

Il n’y a plus qu’à créer une nouvelle recette et tester que le mail est correctement généré.

## Une image contenant texte Description générée automatiquement

## Bonus à prévoir - le composant de mail, le gestionnaire d’événements (Single Responsability et Open Close)