

MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER-PROJET

*Nom de naissance*

- ▶ BASTIDE

*Nom d'usage*

- ▶ BASTIDE

*Prénom*

- ▶ Jordy

*Adresse*

- ▶ 126 Boulevard LAFAYETTE 63000 Clermont-Ferrand

## Titre professionnel visé

**Concepteur développeur d'applications - Niveau II**



# Sommaire

---

<b>Remerciements</b>	<b>4</b>
<b>Résumé du projet en anglais</b>	<b>5</b>
<b>Liste des compétences du référentiel</b>	<b>6</b>
<b>Cahier des charges ou expressions des besoins de l'application à développer</b>	<b>10</b>
<b>Spécifications fonctionnelles</b>	<b>17</b>
<b>Spécifications techniques</b>	<b>28</b>
<b>Réalisations</b>	<b>33</b>
<b>Conclusions</b>	<b>56</b>

## REMERCIEMENTS

Pour commencer, je tiens à remercier Simplon.co, réseau de fabriques numériques, qui m'a fait confiance depuis Septembre 2022. Merci spécifiquement à l'équipe pédagogique s'occupant de la P#7 CDA de m'avoir recruté.

Un grand merci à Yann Bouilhac et David Rigaudie de m'avoir dirigé et aidé à monter rapidement en compétences ainsi qu'à Loïc-Marie Dalmas Reina de m'avoir suivi pendant la durée de mon contrat de professionnalisation.

Je remercie encore toute l'équipe, qui a eu confiance en moi et m'a permis d'atteindre mon objectif et devenir concepteur développeur d'applications. Merci à eux de m'avoir partagé leurs connaissances et fait en sorte que l'ensemble de la formation se déroule le mieux possible.

Je tiens aussi à remercier Nicolas Amblard d'avoir accepté de me prendre en alternance dans son entreprise et me permettre ainsi d'acquérir une réelle expérience en milieu professionnel.

Merci à tous les apprenants Simploniens d'avoir œuvré au bon déroulement de la formation, tant dans sa bonne ambiance que son sérieux quand nécessaire et qui m'ont permis de progresser.

## RÉSUMÉ DU PROJET EN ANGLAIS

Semi Croustillant is a web app which allows users to post recipes.

It's what I like to call it a "2.0 Marmiton" but way more similar to a social media type of platform.

There are three types of users for this web application :

- Visitors ( anonymous )
- Registered users
- Administrators

A visitor can access the home page, consult recipes. He can create an account or sign in / log in.

A logged user can post, edit and delete his own recipes.

He can customize his profile, change his avatar and his related informations, delete his account.

An administrator can create, edit and delete categories.

In future versions, users will be able to search for recipes, like and retrieve them in a dedicated section. They will be able to post comments on recipes (with a Google Recaptcha verification).They also will be able to log in with their google account via an OAuth feature and to follow other users.

In order to realize this project, I've decided to develop it with a Symfony API ( which uses Doctrine as ORM ) and Angular as a front framework to consume it. To store my datas, I choose to go for MySQL and phpMyAdmin as an interface. All of these, orchestrated inside a Docker stack.

I choose to include Bootstrap and FontAwesome libraries for visual parts.

As for the styles sheets, I use scss.

## LISTE DES COMPÉTENCES DU RÉFÉRENTIEL

Voici la liste des compétences à valider pour l'obtention du titre Concepteur Développeur d'Applications.

1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité :
  - a. Maquetter une application
  - b. Développer une interface utilisateur de type desktop
  - c. Développer des composants d'accès aux données
  - d. Développer la partie front-end d'une interface utilisateur web
  - e. Développer la partie back-end d'une interface utilisateur web
2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité :
  - a. Concevoir une base de données
  - b. Mettre en place une base de données
  - c. Développer des composants dans le langage d'une base de données
3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité :
  - a. Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
  - b. Concevoir une application
  - c. Développer des composants métiers
  - d. Construire une application organisée en couches
  - e. Développer une application mobile
  - f. Préparer et exécuter les plans de tests d'une application
  - g. Préparer et exécuter le déploiement d'une application

## Gestion de projet

### Gitlab

Pour ce projet, j'ai décidé d'utiliser la plateforme Gitlab. J'ai utilisé cet outil tout au long de la formation et cela me semblait donc cohérent de continuer de travailler avec.

Cet outil permet de versionner son code. Git est indispensable de nos jours pour tous les développeurs puisqu'il permet de sauvegarder différentes versions du code.

En créant un "repository", nous avons la possibilité de créer des branches qui permettent d'isoler le code de chaque nouvelle fonctionnalité. Lorsqu'une fonctionnalité est opérationnelle, nous pouvons la "merge" sur la branche "dev" correspondant à une version fonctionnelle du code mais potentiellement sujette à divers bugs. Une fois les correctifs appliqués, nous pouvons "merge" la branche "dev" sur la branche "main" ou "master" correspondant à la version du site mise en production. Le système de branche est aussi particulièrement efficace lorsque l'on travaille en équipe.

The screenshot shows the Gitlab interface for a repository named 'sc\_v2'. The sidebar on the left contains various project management icons. The main area shows a commit history for the 'dev' branch, with one commit from 'jordybaste' dated 5 days ago. Below the commit history is a table listing files with their last commit details.

Name	Last commit	Last update
dev	[FRONT] category crud	5 days ago
docker	Initial commit	3 weeks ago
.env.dist	Initial commit	3 weeks ago
.gitignore	[BACK] gitignore images uploads	5 days ago
README.md	Initial commit	3 weeks ago
docker-compose.yaml	Initial commit	3 weeks ago

The screenshot shows the GitHub Desktop application interface. The main window displays a list of commits for the 'dev' branch of the repository 'taste4me'. The selected commit is 'gitignore images uploads' (commit b030753). The commit details show it was made by 'jordybastide' on July 11, 2022, at 21:22. The commit message is '[FRONT] category crud'. Below the commit message, there is a file tree showing the changes made to files in the 'dev/app/src/app' directory.

Author	Date	Commit SHA	Message
jordybastide	11 juil. 2022 21:24:26 +02:00	b030753	[FRONT] category crud
jordybastide	11 Jul 2022 21:22	b1f2a6b	[BACK] gitignore images uploads
jordybastide	11 Jul 2022 20:44	d1ded8f	[BACK] fix services.yaml
jordybastide	11 Jul 2022 20:43	e6b51b1	[BACK] fix category controller
jordybastide	11 Jul 2022 20:33	03ef335	[BACK] category controller & uploads path
jordybastide	11 Jul 2022 20:32	a4a61ce	[BACK] category form type
jordybastide	11 Jul 2022 20:32	763b1e0	[BACK] update category entity
jordybastide	10 Jul 2022 17:56	2416a72	[FRONT] add categories crud
jordybastide	10 Jul 2022 17:08	c3586fe	[BACK][FRONT] refactor recipes crud routes
jordybastide	10 Jul 2022 17:08	6d047d1	[BACK] add category entity, controller, form type
jordybastide	10 Jul 2022 14:53	87599de	[FRONT] currentUser
jordybastide	10 Jul 2022 14:40	76640e4	[FRONT] current user to local storage
jordybastide	10 Jul 2022 14:40	154c168	[BACK] api login
jordybastide	10 Jul 2022 14:38	16c2632	[BACK] token generator
jordybastide	10 Jul 2022 14:37	7ffe250	[BACK] added user fields
jordybastide	8 Jul 2022 15:30	2b9cf99	[BACK] api login controller
jordybastide	8 Jul 2022 15:30	d08e3d1	[BACK] Token generator
jordybastide	8 Jul 2022 15:29	6a8c0c7	[FRONT] refactoring file tree

Etant donné que j'étais seul à travailler sur le repository, je n'ai pas créé de branche à chaque nouvelle fonctionnalité ( processus standard que nous suivions en entreprise ) mais j'ai développé uniquement sur la branche "dev".

## Trello

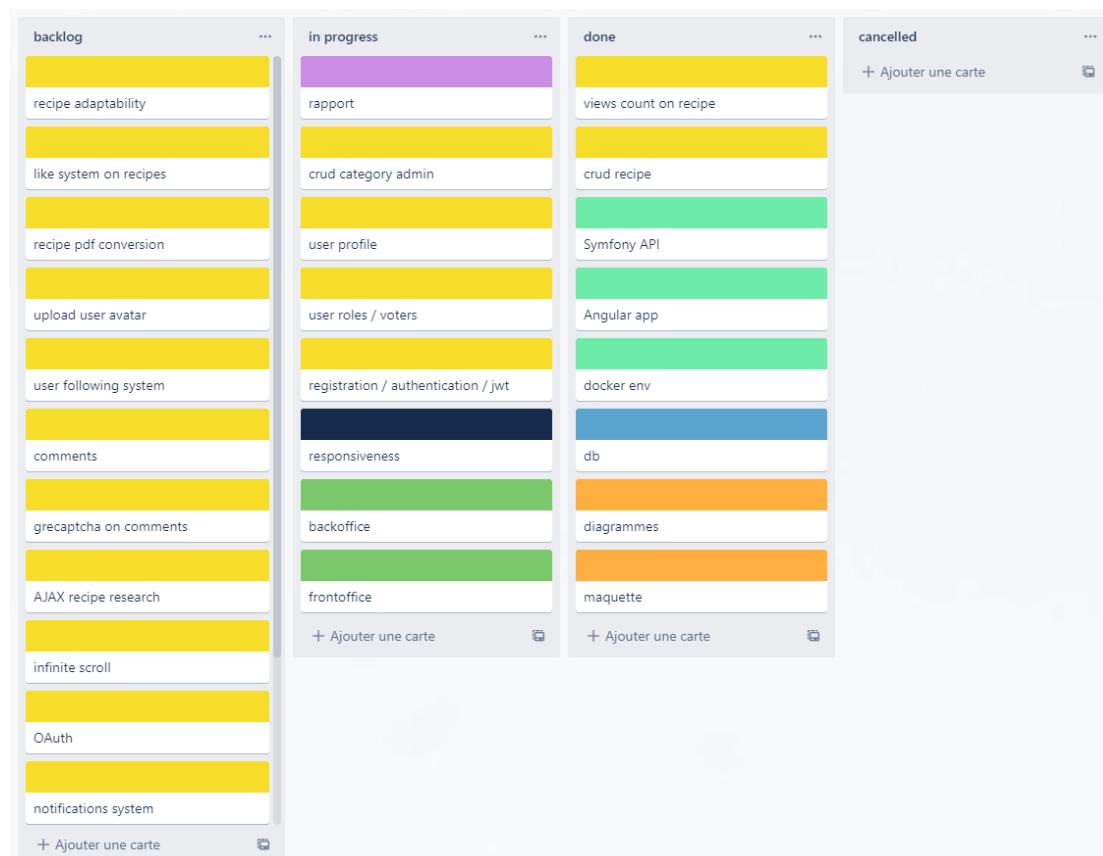
J'ai aussi fait le choix d'utiliser Trello afin de m'organiser, grâce à leur système de tableau me permettant de créer des tâches à effectuer. Je peux déplacer

une tâche dans la colonne correspondante. J'ai estimé que 4 colonnes suffiraient pour ce projet :

- “backlog” qui référence l'ensemble des tâches à exécuter
- “in progress” pour les tâches en cours
- “done” pour les tâches terminées
- “canceled” pour les tâches annulées

Je n'ai pas spécialement déterminé la durée de mes sprints en amont car je préférais aller à mon rythme et ne pas me mettre trop de “pression”.

J'ai fait le choix de mettre en place un système de code couleur pour chaque tâche afin de m'y retrouver plus facilement. Par exemple, en jaune nous avons les fonctionnalités, en vert des tâches plus générales, en rose les tâches optionnelles et cetera...



## CAHIER DES CHARGES OU EXPRESSIONS DES BESOINS DE L'APPLICATION À DÉVELOPPER

### Contexte

N'ayant pas de réel client pour ce projet, j'ai décidé de refaire le projet présenté lors de ma soutenance de Dev web.

Semi-croustillant permet aux utilisateurs de partager des recettes sur la plateforme.

C'est une app développée en Angular qui consomme une API Symfony, le tout dans une stack Docker.

Les utilisateurs pourront créer un compte, poster des recettes, enregistrer des recettes qui leur plaisent, laisser des commentaires et suivre d'autres utilisateurs.

Cette application remplira les critères de compétences requises pour l'obtention du titre professionnel, à savoir :

- Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité :
  - Maquetter une application
  - Développer des composants d'accès aux données
  - Développer la partie front-end d'une interface utilisateur web
  - Développer la partie back-end d'une interface utilisateur web
- Concevoir et développer la persistance des données en intégrant les recommandations de sécurité :
  - Concevoir une base de données
  - Mettre en place une base de données

- Développer des composants dans le langage d'une base de données
- Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité :
  - Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
  - Concevoir une application
  - Développer des composants métier
  - Construire une application organisée en couches
  - Préparer et exécuter les plans de tests d'une application
  - Préparer et exécuter le déploiement d'une application

## Gestion des utilisateurs

Pour cette première version, il y a quatre types d'utilisateurs :

- le visiteur (anonyme)
- l'utilisateur connecté
- l'administrateur

Le visiteur (non connecté) pourra accéder aux recettes, à la recherche de recettes et poster des commentaires sous condition de validation d'un Google ReCaptcha. Il pourra également créer un compte ou se connecter avec un compte Google afin de devenir un utilisateur enregistré.

L'utilisateur enregistré a les mêmes droits que le visiteur mais pourra en plus, poster des recettes, les éditer et les supprimer. Il pourra aussi modifier son profil, son avatar, les informations qui y sont relatives et évidemment supprimer son compte s'il le souhaite. Il aura également la possibilité d'enregistrer des recettes, suivre d'autres utilisateurs et poster des commentaires.

Un administrateur peut ajouter, éditer ou supprimer une catégorie.

Pour la version suivante, j'envisage de mettre en place un système de notifications pour les utilisateurs et une interface administrateur plus complète, permettant notamment de modérer les commentaires. Aussi, je prévois de pouvoir partager une recette sur différents réseaux sociaux et d'inclure un formulaire de contact pour des suggestions, comme par exemple la création d'une catégorie.

## Fonctionnalités

### A. V1 :

- Inscription
- Connexion
- Consulter les recettes
- Rechercher les recettes
- “Exporter” une recette en PDF
- Imprimer une recette
- Consulter les catégories et les recettes associées
- Modifier son profil
- Modifier son avatar
- Modifier son mot de passe
- Supprimer son compte
- Poster une recette
- Modifier une recette
- Supprimer une recette
- Un administrateur peut créer / éditer / supprimer une catégorie

### B. V2 :

- Poster des commentaires
- Inscription et Connexion avec un compte Google ( Oauth )
- “Liker” les recettes des autres utilisateurs et les enregistrer
- Accéder au profil d'un utilisateur et consulter ses recettes
- Suivre un utilisateur

- Scroll infini sur les liste de recettes
- Adaptabilité des recettes au nombre de personnes
- “Ingredients reminder”

## Inscription

*Accessible pour : le visiteur*

Permet de créer un compte en rentrant son email, son prénom, son nom, son pseudonyme et son mot de passe.

## Connexion

*Accessible pour : le visiteur et l'utilisateur enregistré*

Une fois le compte créé, permet de s'identifier pour profiter de toutes les fonctionnalités de la plateforme.

## Consulter les recettes

*Accessible pour : le visiteur et l'utilisateur enregistré*

Permet d'accéder à toutes les recettes et de cliquer dessus pour en consulter le contenu. Peut aussi exporter la recette en PDF et l'imprimer.

## Rechercher les recettes

*Accessible pour : le visiteur et l'utilisateur enregistré*

Permet de rechercher une recette par mots clés de façon asynchrone. (Limitation du nombre de requêtes pour ne pas “flood” le serveur)

## “Exporter” une recette en PDF

*Accessible pour : le visiteur et l'utilisateur enregistré*

Permet à l'utilisateur d'exporter une recette sous forme de PDF

## Imprimer une recette

*Accessible pour : le visiteur et l'utilisateur enregistré*

Permet à l'utilisateur d'imprimer une recette ayant un aspect spécifique à l'impression.

**Consulter les catégories et les recettes associées**

*Accessible pour : le visiteur et l'utilisateur enregistré*

Permet à l'utilisateur de consulter les catégories et les recettes qui y sont associées.

**Modifier son profil**

*Accessible pour : l'utilisateur enregistré*

Permet de modifier l'ensemble des informations de son profil ( pseudonyme, nom, prénom et cetera...) Un utilisateur ne pourra pas modifier les informations d'un autre utilisateur.

**Modifier son avatar**

*Accessible pour : l'utilisateur enregistré*

Permet à l'utilisateur "d'uploader" une image pour son avatar et ainsi changer celle par défaut.

**Modifier son mot de passe**

*Accessible pour : l'utilisateur enregistré*

Permet de modifier le mot de passe existant. Un lien de réinitialisation est envoyé à l'utilisateur par email et il doit cliquer sur un lien afin de rentrer son nouveau mot de passe (expiration du lien au bout d'une heure).

**Supprimer son compte**

*Accessible pour : l'utilisateur enregistré*

Permet à l'utilisateur de supprimer son compte s'il le souhaite. Cette action entraînera la suppression de toutes les données associées à l'utilisateur.

**Poster une recette**

*Accessible pour : l'utilisateur enregistré*

Permet de créer une recette en suivant les étapes (titre de la recette, durée, difficulté, nombre de personnes, ajout d'ingrédients de manière dynamique, définition des étapes de préparation et cuisson, upload d'image "thumbnail" ).

**Modifier une recette**

*Accessible pour : l'utilisateur enregistré*

Permet de modifier une recette existante. Un utilisateur ne pourra pas modifier les recettes d'un autre utilisateur.

**Supprimer une recette**

*Accessible pour : l'utilisateur enregistré*

Permet de supprimer une recette. Un utilisateur ne pourra pas supprimer les recettes d'un autre utilisateur.

**Poster des commentaires**

*Accessible pour : le visiteur et l'utilisateur enregistré*

Permet de poster un commentaire sous une recette. L'utilisateur devra valider un Google ReCaptcha afin d'avoir un minimum de sécurité.

**Inscription et Connexion avec Google**

*Accessible pour : le visiteur*

Permet de s'authentifier avec Google Oauth si l'utilisateur dispose d'un compte Google.

**“Liker” les recettes des autres utilisateurs et les enregistrer**

*Accessible pour : l'utilisateur enregistré*

Permet de “liker” une recette et de la retrouver dans la rubrique “Mes recettes enregistrées”

**Accéder au profil d'un utilisateur et consulter ses recettes**

*Accessible pour : le visiteur et l'utilisateur enregistré*

Permet d'accéder au profil d'un utilisateur et de consulter les recettes qu'il a publié ( à la manière d'un profil Instagram ).

**Suivre un utilisateur**

*Accessible pour : l'utilisateur enregistré*

Permet de suivre un utilisateur.

## **Création, édition et suppression de catégories**

*Accessible pour : un administrateur uniquement*

La gestion des catégories est uniquement réservée à l'administrateur pour le moment.

## **“Scroll infini”**

*Accessible pour : le visiteur et l'utilisateur enregistré*

Cette fonctionnalité asynchrone vient remplacer la pagination classique et donne à l'application un aspect plus moderne et orientée mobile.

## **Adaptabilité des recettes**

*Accessible pour : le visiteur et l'utilisateur enregistré*

Cette fonctionnalité permet de changer le nombre de personnes et d'ajuster dynamiquement les quantités des ingrédients d'une recette.

## **“Ingredients reminder”**

*Accessible pour : le visiteur et l'utilisateur enregistré*

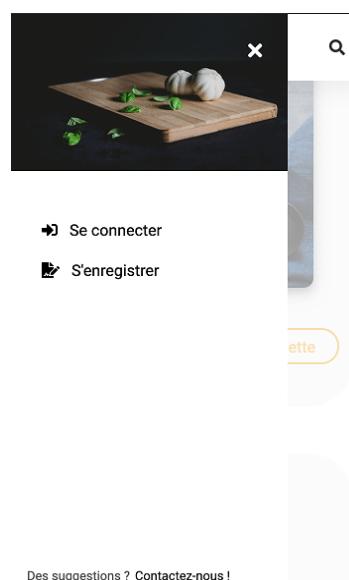
Permet d'afficher les ingrédients et leur quantité respective tout au long des étapes de la recette.

## SPÉCIFICATIONS FONCTIONNELLES

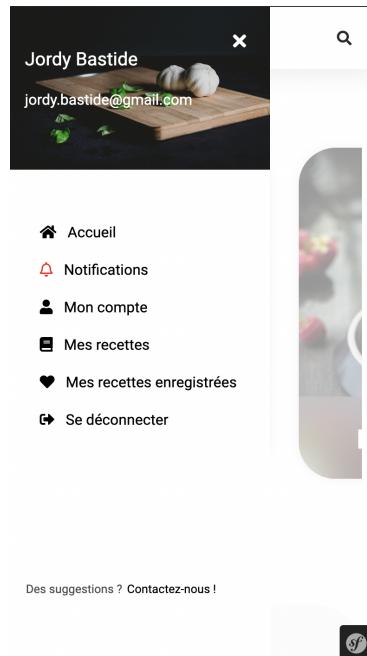
### Use case

Dans cette première version de l'application web, j'ai décidé d'avoir 3 types d'utilisateurs : le visiteur (anonyme), l'utilisateur connecté et l'administrateur. Le rôle d'administrateur sera agrémenté dans les futures versions, notamment avec une modération de commentaires.

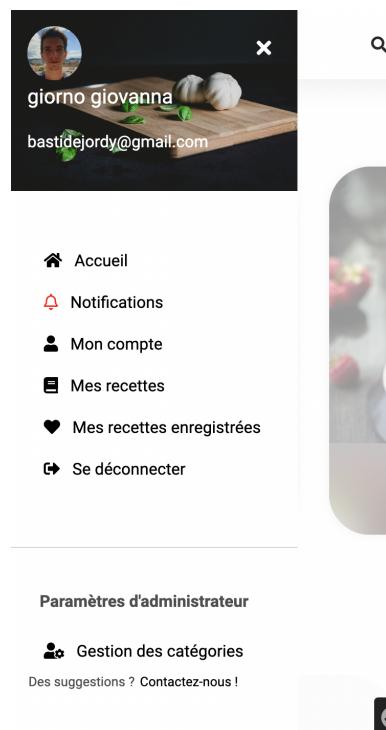
Grâce à la définition de ces différents rôles, cela me permet de contrôler les accès des utilisateurs à certaines fonctionnalités. Par exemple, un visiteur n'a pas accès à la partie "Mon compte" ou "Mes recettes". Un utilisateur enregistré n'aura pas accès non plus à la gestion de catégories que je souhaite déléguer à l'administrateur.



*Ci-dessus, le menu ouvert sans être connecté.*



*Ci-dessus, le menu ouvert une fois que l'utilisateur est authentifié.*



*Ci-dessus, le menu ouvert en tant qu'administrateur.*

## Conception

### Diagramme de base de données

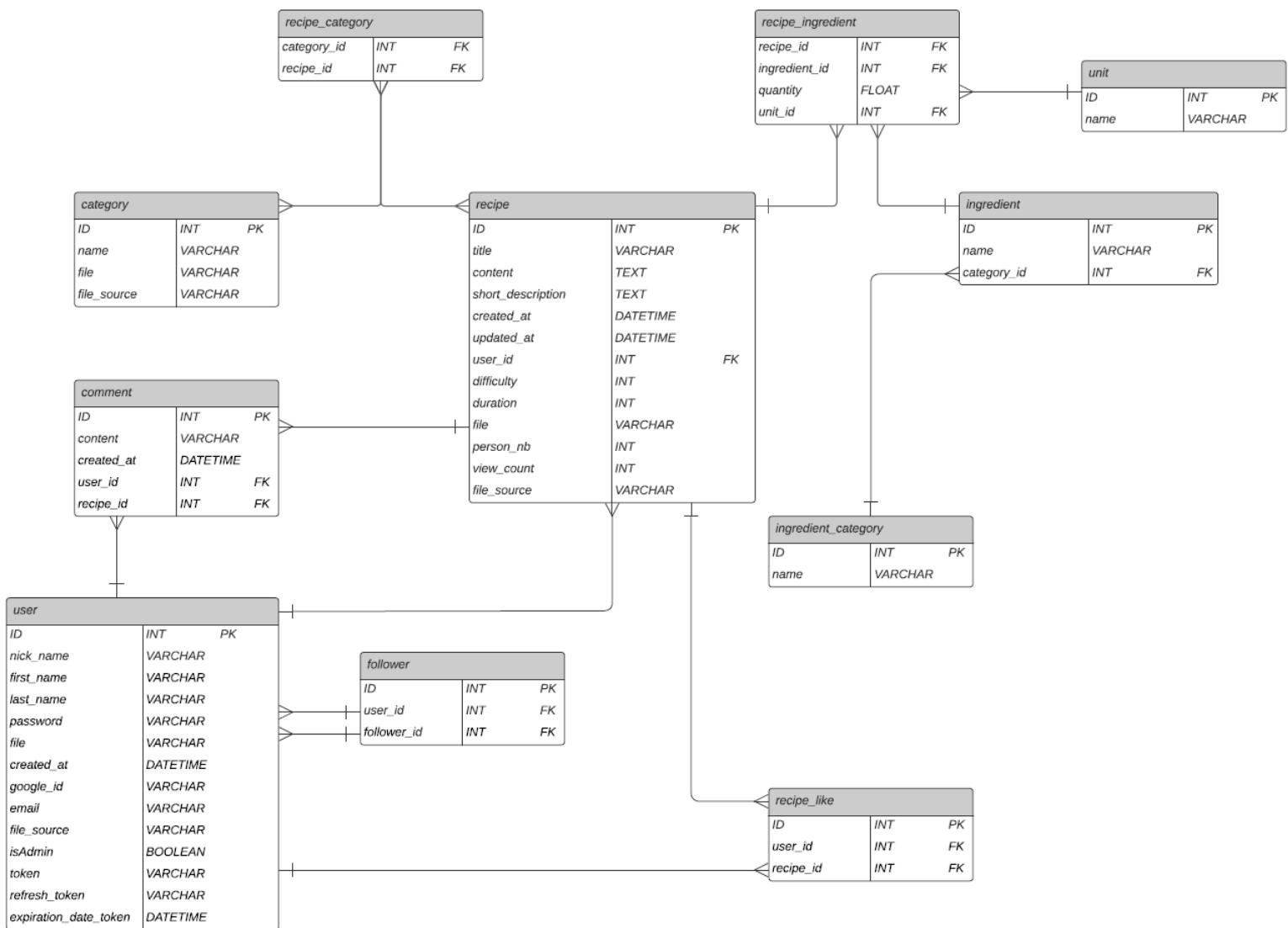
Toujours pendant l'étape de conception, j'ai réalisé un schéma de base de données. Il est composé des différentes tables qui représentent mes modèles. A l'intérieur de celles-ci, on retrouve les colonnes, qui sont en somme les propriétés de mes modèles. La plupart des tables comportent un champ "id" qui est une clé primaire et qui s'auto-incrémente. Ainsi, chaque instance de mon modèle est atteignable via son "id". Dans ce diagramme, on peut aussi voir les relations entre les tables.

Dans un modèle de base de données dit "relationnel", chaque colonne peut avoir une relation avec une colonne d'une autre table. Il existe plusieurs types de relations :

- One-to-One : A peut avoir un B
- One-to-Many : A peut avoir plusieurs B (dans mon cas, une catégorie d'ingrédients peut avoir plusieurs ingrédients )
- Many-to-One : Plusieurs A peuvent avoir un B
- Many-to-Many : Plusieurs A peuvent avoir plusieurs B (dans mon cas, une recette peut se retrouver dans plusieurs catégories, et une catégorie peut contenir plusieurs recettes )

Dans ce dernier cas, on utilise une table de jointure ( ou relationnelle ), c'est-à-dire une table intermédiaire qui comportera dans la plupart des cas deux colonnes : A\_id et B\_id qui font donc référence à nos deux entités et qui combinées, forment une clé primaire. Les relations se font à l'aide de "Foreign key" (ou clé étrangères qui permettent d'avoir des relations propres et non redondantes notamment grâce à des contraintes).

J'ai utilisé le modèle relationnel car je le trouve plus approprié et étant donné que nous n'avons utilisé que ce modèle en formation, il me semblait plus pertinent.



Ci-dessus, le MCD réalisé sur Lucidchart.

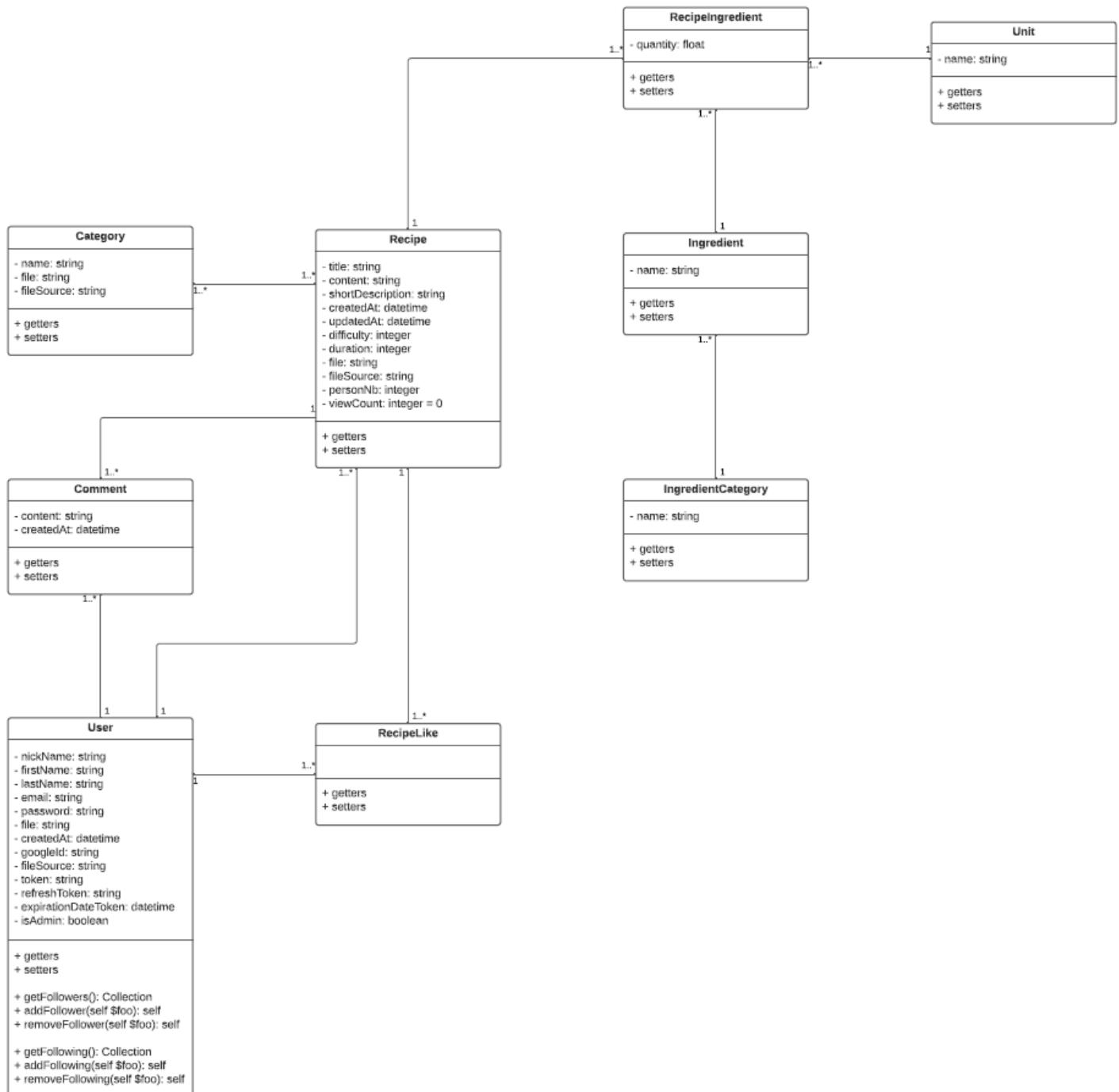
## Diagramme de classes

Les diagrammes de classes décrivent clairement la structure d'un système particulier en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets.

Ils permettent de mieux comprendre l'aperçu général des schémas d'une

Titre Professionnel Concepteur Développeur d'Applications

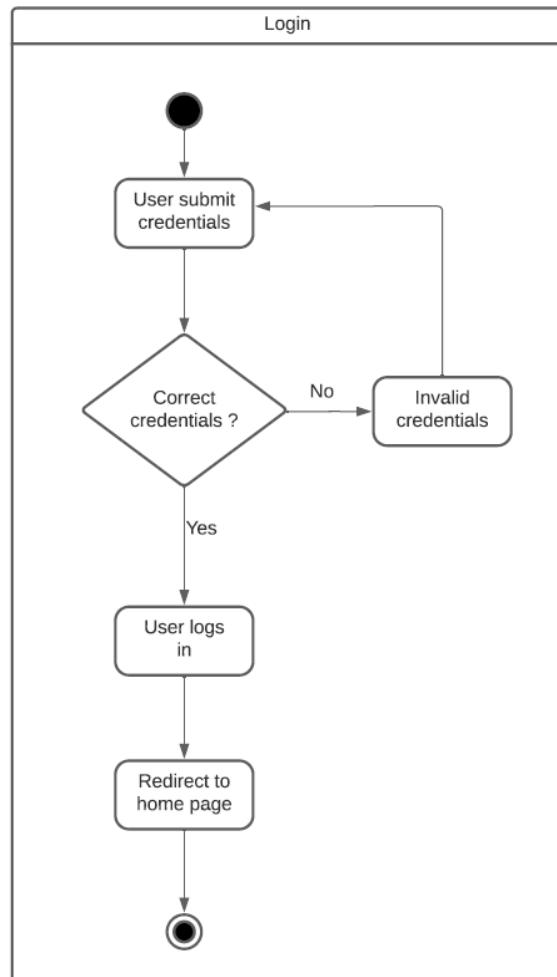
application et donc permettent de mettre l'accent sur ce qui doit être programmé et comment cela doit être programmé.



Ci-dessus, le diagramme de classes réalisé sur Lucidchart.

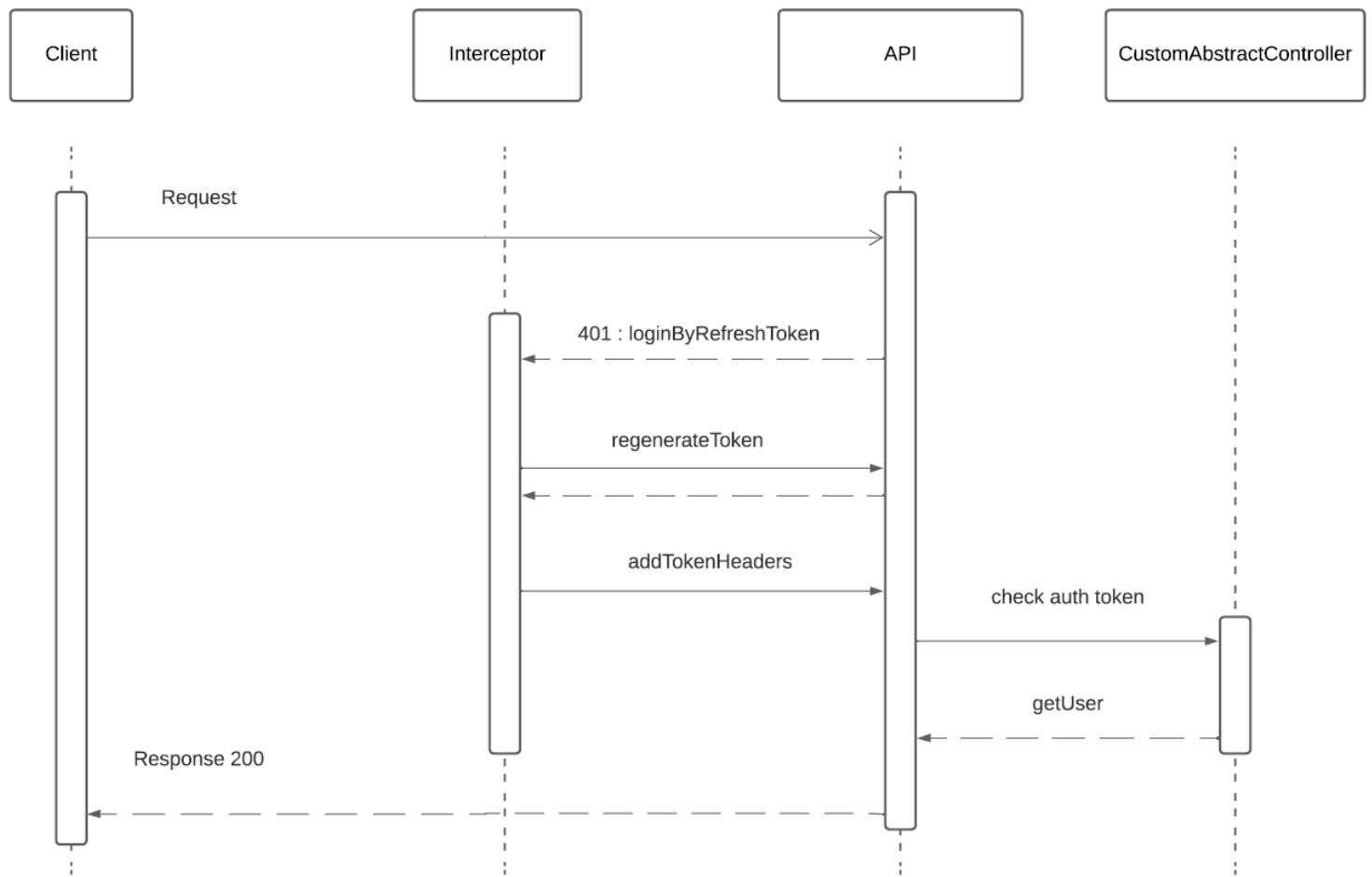
## Autres Diagrammes UML

### 1. Diagramme d'activité



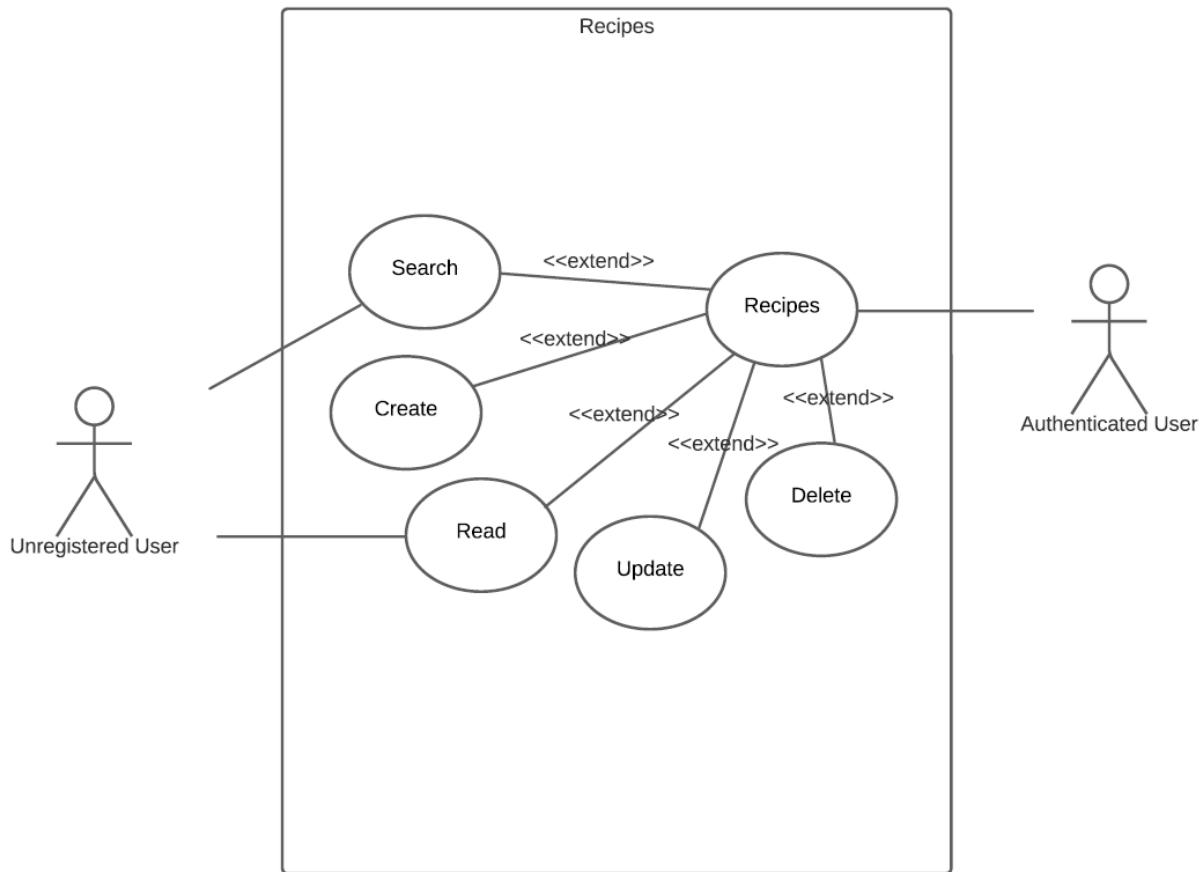
Ci-dessus, le diagramme d'activité réalisé sur Lucidchart.

### 2. Diagramme de séquence



Ci-dessus, le diagramme de séquence réalisé sur Lucidchart.

### 3. Diagramme use case



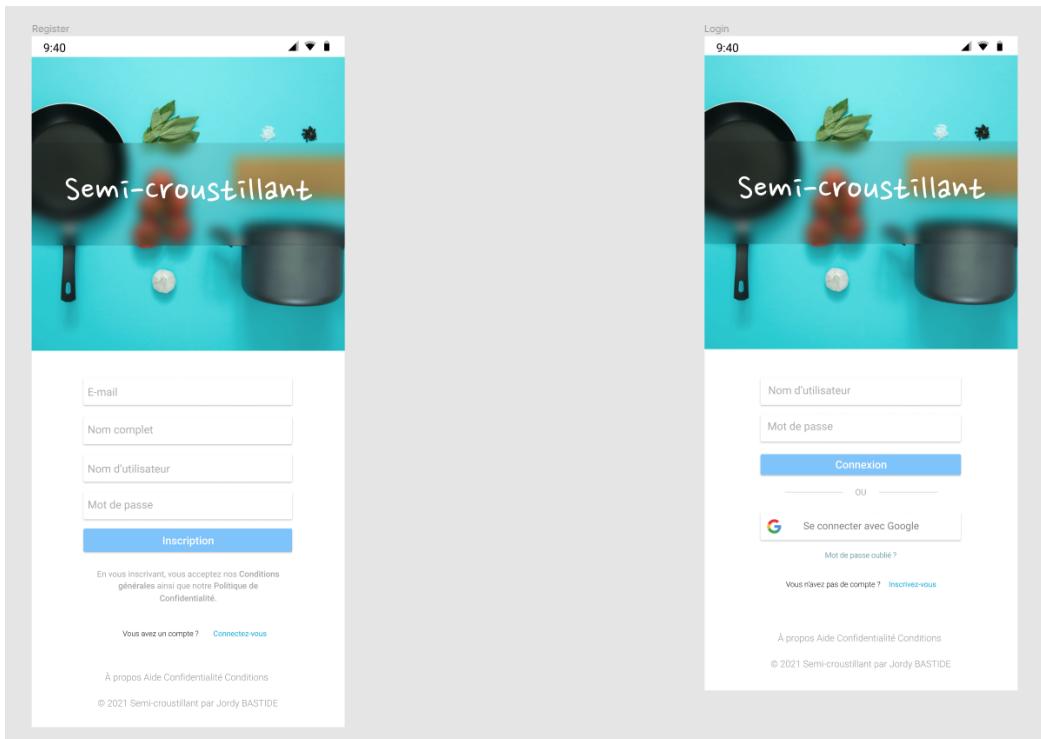
*Ci-dessus, le diagramme use case réalisé sur Lucidchart.*

## Maquette

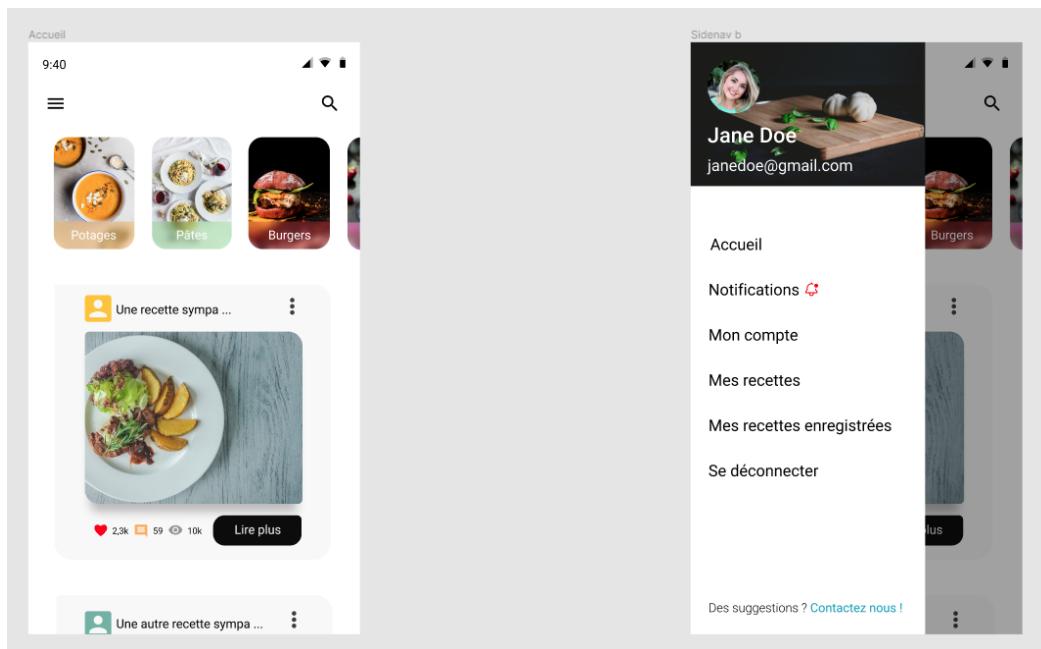
A partir du wireframe, il m'a été plus simple de réaliser les maquettes graphiques, permettant d'obtenir un résultat visuel plus proche de l'aspect final de l'application web.

J'ai fait le choix de partir sur un design simple inspiré de Material Design ( norme Google ) afin de garder un ensemble cohérent et ludique autant pour

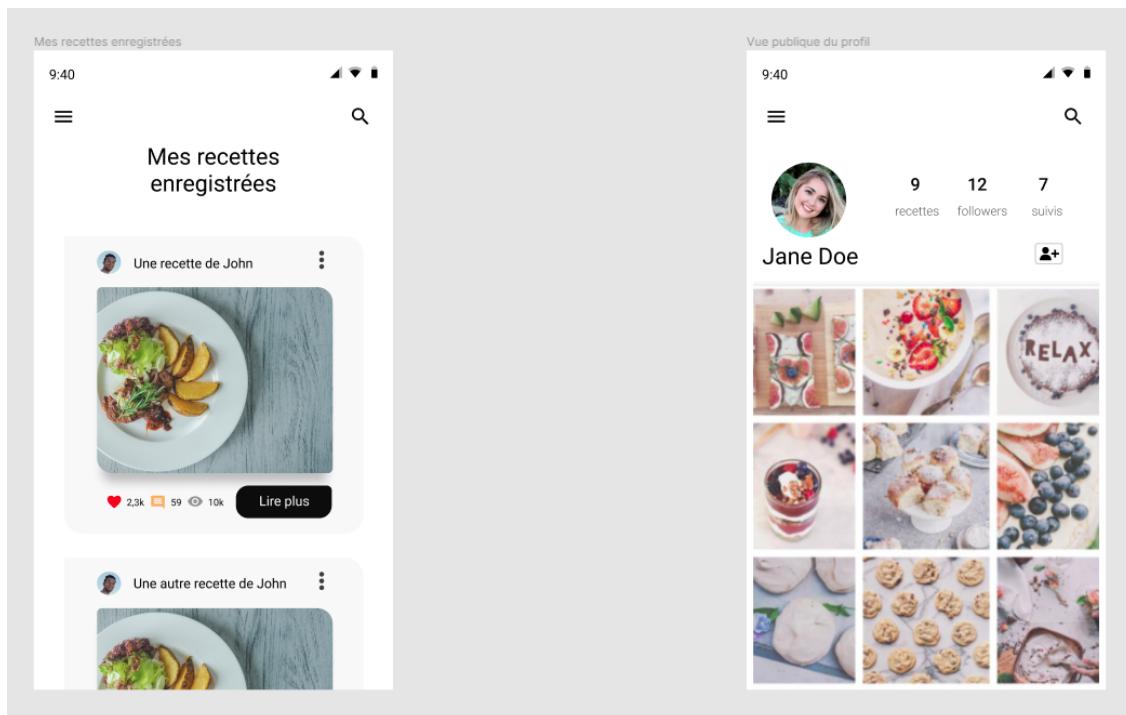
l'aspect UX ( expérience utilisateur ) que pour l'aspect UI ( interface utilisateur ). Je suis parti sur une approche Mobile First car je me suis habitué à travailler les maquettes de cette façon le plus tôt possible dans la formation. J'ai utilisé le logiciel Figma pour les réaliser.



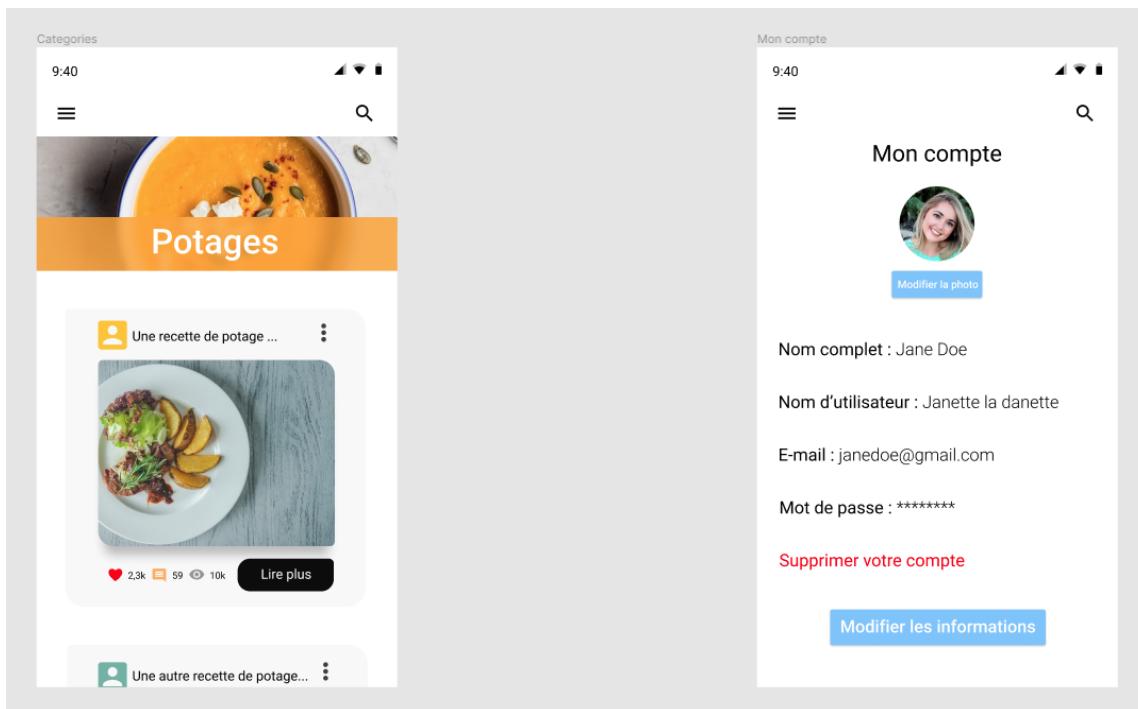
*Ci-dessus, l'inscription à gauche et la connexion à droite.*



*Ci-dessus, l'accueil à gauche et la sidenav à droite.*



*Ci-dessus, les recettes enregistrées à gauche et le profil d'un utilisateur à droite.*



Ci-dessus, la vue d'une catégorie à gauche et la modification du profil à droite.

## SPÉCIFICATIONS TECHNIQUES

### Stack globale

#### Docker

Docker est une plateforme permettant de lancer certaines applications dans des conteneurs logiciels.

Il permet d'orchestrer des volumes au sein d'un même “container”.

C'est aussi un outil très pratique puisqu'il permet de “pull” les images depuis le DockerHub et de choisir n'importe quelle version sans que l'on ait ces versions installées sur notre machine.

Dans le cadre de ce projet, j'ai par exemple un node 14 alpine et un php 7.4 fpm alpine.

fpm : Fast Process Manager, permet d'exécuter php  
alpine: version allégée

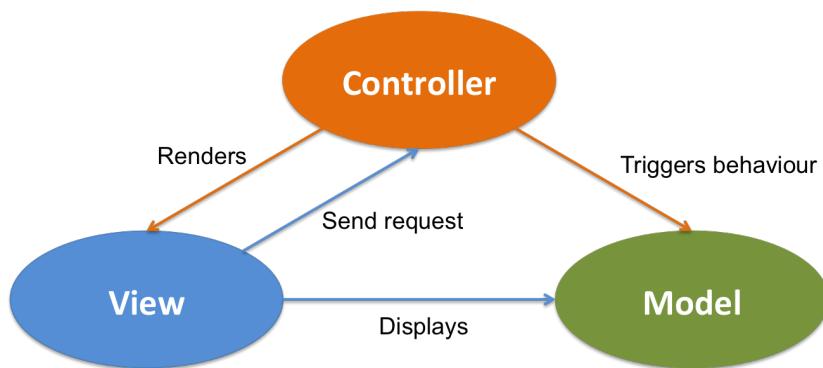
#### Back-end

##### Symfony API et Doctrine

Symfony est un framework qui repose sur le design pattern MVC (“Model Vue Controller”) :

- Le Model : C'est la définition de la structure des entités de l'application (par exemple : un User, une Recipe ou un Message). Par convention, un nom de modèle prend toujours une majuscule. On y définit également ses relations avec les autres entités.

- La Vue : C'est l'affichage des données envoyées par le Controller. Symfony utilise le moteur de templating Twig par défaut mais dans notre cas, nous utiliserons Angular pour consommer les données de notre API.
- Le Controller : C'est la partie qui contient toute la logique et les actions. Ainsi je vais pouvoir retourner les données désirées à l'appel d'une route grâce à une fonction d'un Controller.



Le modèle MVC a pour principal avantage de diviser le code. Il est, par conséquent, plus facile de localiser les bugs et d'éviter de réécrire du code existant. Cette efficacité fait du modèle MVC le design pattern le plus utilisé dans le web actuellement.

Symfony utilise l'ORM (“Object-Relational Mapping”) Doctrine. Un ORM est une interface qui permet d'interagir avec la base de données sans directement écrire du SQL, dans le cas d'une base MySQL par exemple. Ainsi, de simples fonctions PHP permettent de faire des requêtes dans la base de données de façon sécurisée. Doctrine permet de faire le lien entre nos entités dans le code et nos entités en base de données.

Dans le cadre de ce projet, j'utilise uniquement Symfony sous forme d'API. Je n'utilise pas le moteur de templating Twig.

Une API (application programming interface ou « interface de programmation

d'application ») est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

## La base de données

En ce qui concerne la base de données, j'ai choisi une base de données relationnelle : MySQL. Elle a l'avantage d'être gratuite et universelle. La majorité des hébergeurs web l'utilise. De plus, c'est le seul type de base de données que j'ai vu en formation, c'est donc naturellement que je me suis tourné vers ce choix.

## Front-end

### Angular

Angular est un framework côté client, open source, basé sur TypeScript. Dans le cadre de ce projet, il va permettre de consommer notre API.

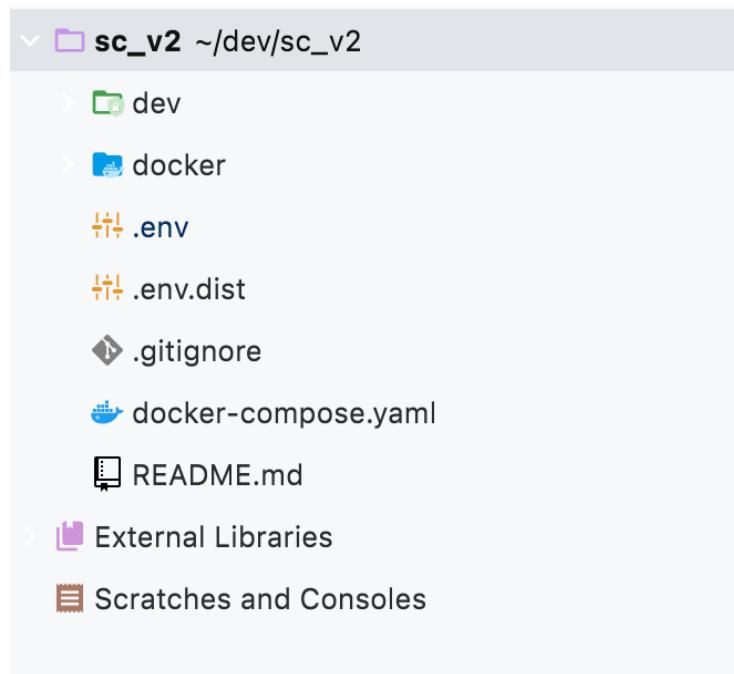
### Bootstrap

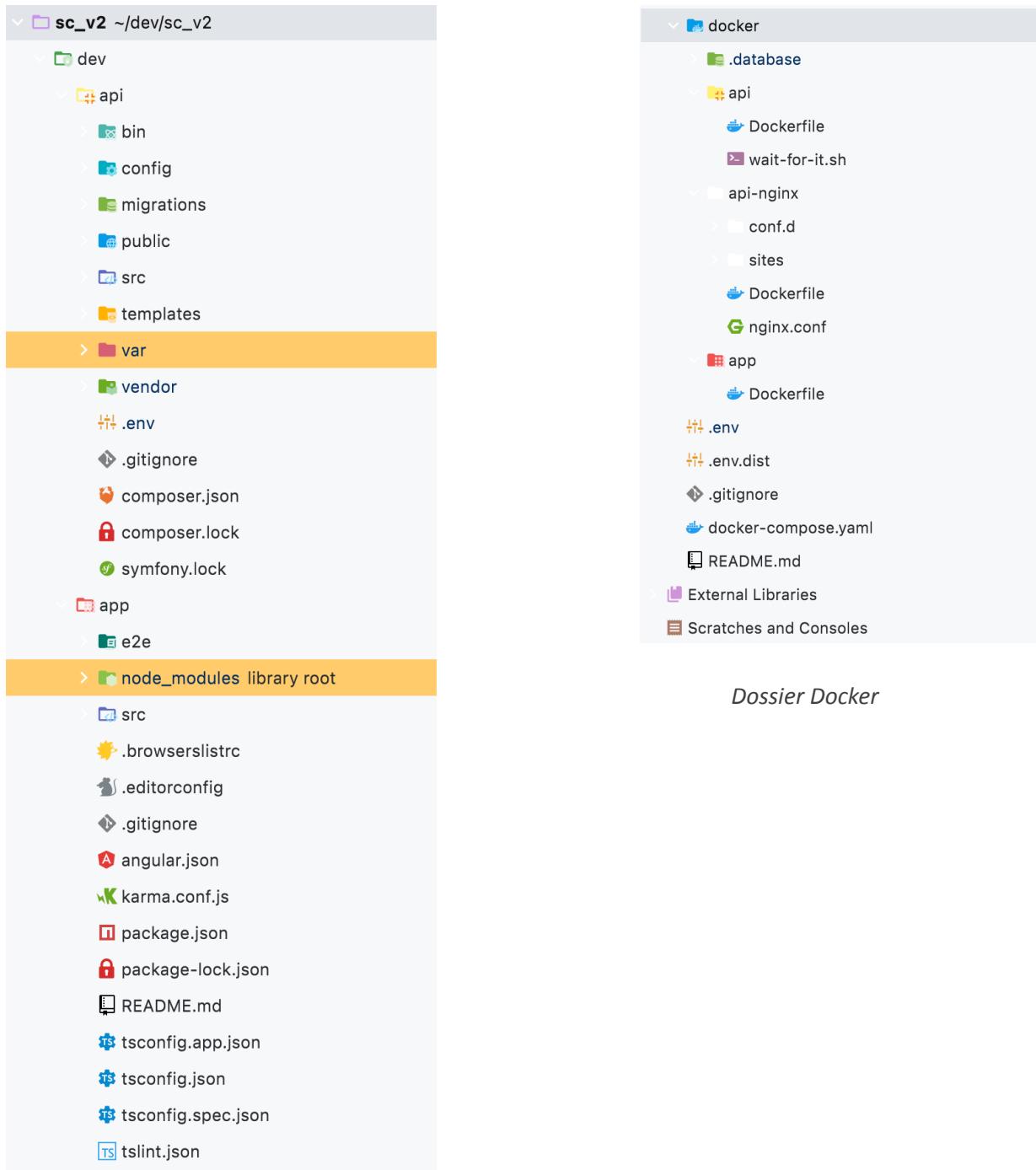
J'ai choisi d'utiliser cette librairie car je souhaitais un design simple et dans l'ère du temps. Il suffit d'ajouter des classes CSS sur une balise HTML afin d'en activer les propriétés.

Je justifie aussi ce choix car j'ai beaucoup utilisé cette librairie lors de mon alternance et elle me permet de gagner beaucoup de temps.

J'ai aussi ajouté du CSS "custom" pour personnaliser mon application web afin qu'elle se démarque de l'aspect des composants Bootstrap par défaut.

## Structure du dossier





*Dossier Dev  
( contenant Symfony API et Angular App )*

## RÉALISATIONS

# Développement

## Installation

docker-compose.yml :

```
1  version: '3'
2
3  services:
4    database:
5      container_name: sc-mysql
6      platform: linux/x86_64
7      image: mysql:5.7
8      volumes:
9        - ./docker/.database/data:/var/lib/mysql
10     ports:
11       - "3307:3306"
12     restart: always
13     environment:
14       MYSQL_ROOT_PASSWORD: ${DATABASE_PASSWORD}
15       MYSQL_DATABASE: ${DATABASE_NAME}
16       MYSQL_USER: ${DATABASE_USER}
17       MYSQL_PASSWORD: ${DATABASE_PASSWORD}
18
19    api:
20      container_name: sc-api
21      build:
22        context: ./docker/api
23      depends_on:
24        - database
25      environment:
26        - APP_ENV=${APP_ENV}
27        - APP_SECRET=${APP_SECRET}
28        - DATABASE_URL=mysql://${DATABASE_USER}:${DATABASE_PASSWORD}@database:3306/${DATABASE_NAME}?serverVersion=5.7
29      volumes:
30        - ./dev/api/:/var/www
31
```

```

32 ► app:
33   container_name: sc-app
34   build:
35     context: ./docker/app
36     target: dev-stage
37   volumes:
38     - ./dev/app/:/app
39   ports:
40     - "80:4200"
41
42 ► api-nginx:
43   container_name: sc-nginx
44   build:
45     context: ./docker/api-nginx
46   volumes:
47     - ./dev/api/:/var/www
48     - ./docker/api-nginx/nginx.conf:/etc/nginx/nginx.conf
49     - ./docker/api-nginx/sites/:/etc/nginx/sites-available
50     - ./docker/api-nginx/conf.d:/etc/nginx/conf.d
51     # - ./docker/logs:/var/log
52   depends_on:
53     - api
54   ports:
55     - "81:80"
56
57 ► phpmyadmin:
58   container_name: sc-phpmyadmin
59   image: phpmyadmin/phpmyadmin
60   restart: always
61   ports:
62     - "8081:80"
63   environment:
64     PMA_HOST: database
65     MYSQL_ROOT_PASSWORD: ${DATABASE_PASSWORD}
66   UPLOAD_LIMIT: 300M

```

Dockerfile pour les images :

PHP:

```

1 ► FROM php:7.4.15-fpm-alpine
2 COPY wait-for-it.sh /usr/bin/wait-for-it
3 RUN chmod +x /usr/bin/wait-for-it
4 RUN apk --update --no-cache add git
5 RUN docker-php-ext-install pdo_mysql
6 COPY --from=composer /usr/bin/composer /usr/bin/composer
7 WORKDIR /var/www
8 CMD composer install ; wait-for-it database:3307 -- bin/console doctrine:migrations:migrate ; php-fpm
9 EXPOSE 9000

```

## Node:

```

1 ►> FROM node:14-alpine AS dev-stage
2 WORKDIR /app
3 RUN npm install -g @angular/cli
4 CMD ["ng", "serve", "--host=0.0.0.0"]
5 #ng serve
6
7 FROM dev-stage AS build-stage
8 WORKDIR /app
9 COPY ./ .
10 RUN npm ci #plus safe que npm install (prend en compte les versions)
11 RUN ng build sc-app
12
13 FROM nginx:alpine AS production-stage
14 WORKDIR /usr/share/nginx/html
15 COPY --from=build-stage /app/dist/sc-app .

```

## php bin/console - Entities et Repositories

J'ai ensuite entamé la création de mes entités tout en suivant mon Model Conceptuel de Données à l'aide des commandes suivantes :

**docker exec -it sc-api sh**

*Permet de rentrer dans notre volume Symfony en mode bash pour lancer les commandes Symfony*

**php bin/console make:entity**

*Permet de créer / modifier une entité*

Admettons que je souhaite créer mon entité Recette. Lorsque je tape cette commande, Symfony me guide tout au long du processus en commençant par le choix du nom de l'entité ( ici, Recipe ). Une fois choisi, je peux ajouter des propriétés à cette entité. L'ajout d'une propriété se découpe en plusieurs étapes et se déroule toujours de la manière suivante :

- choix du nom de la propriété ( par exemple “title” )
- choix du type de la donnée ( dans mon cas, une string et c'est aussi à ce moment que l'on peut choisir du type de relation si nous souhaitons lier notre entité à une autre)
- choix de la longueur ( ici 255, étant la valeur suggérée par défaut )
- la donnée est-elle “nullable” ? ( dans mon cas, je ne souhaite pas de titre nullable sur une recette )

A l'issue de ces étapes, Symfony me génère les fichiers suivants :

- src/Entity/Recipe.php
- src/Repository/RecipeRepository.php

Lorsque j'ai terminé de créer mes entités, j'utilise les commandes suivantes pour générer la base de données et insérer mes entités et leurs propriétés :

- **php bin/console doctrine:database:create** ( crée la base de données )
- **php bin/console make:migration** (un fichier de migration est créé dans le dossier migrations du projet et il contient les requêtes SQL nécessaires pour mettre à jour la base de données )
- **php bin/console doctrine:migrations:migrate** ( exécute le fichier de migration )

Les migrations permettent de “rollback” ( de revenir en arrière ) sans avoir à tout effacer et recréer si l'on souhaite faire des modifications sur certaines entités.

## **php bin/console - Les Controllers**

Pour générer les controllers, il existe une commande Symfony :

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

La particularité de cette commande est qu'elle va créer mes controllers avec les méthodes CRUD ( create read update delete ) ainsi que les vues Twig associées à ces dernières.

Cependant dans notre cas, nous devons communiquer avec notre application Angular; de ce fait, nous devons sérialiser nos objets PHP et les manipuler au format JSON.

De plus, nous n'avons pas non plus besoin des vues Twig. J'ai donc dû écrire le contenu des controllers manuellement.

## Registration / Authentication

Après avoir généré mon entité User, il me semblait logique de créer mon système d'enregistrement d'utilisateur, d'authentification et de réinitialisation de mot de passe.

Il y a deux façons de s'authentifier sur l'application, par email / mot de passe ou via refreshToken.

J'ai créé tout le système, je n'ai pas utilisé de plugin / librairie / bunde.

En plus de l'authentification, les tokens vont me permettre de vérifier l'utilisateur courant et de vérifier que ce dernier a bien les droits d'accéder à la ressource ciblée.

Pour cela j'utilise un interceptor côté client :

```

25  ƒ intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<object>> {
26    let authReq = req;
27    const userLocalStorage = localStorage.getItem(key: 'currentUser');
28    // @ts-ignore
29    const user = JSON.parse(userLocalStorage);
30    let token = null;
31    if (user) {
32      token = user.token;
33    }
34    if (token != null) {
35      authReq = this.addTokenHeader(req, token);
36    }
37    // @ts-ignore
38    return next.handle(authReq).pipe(catchError(selector: error => {
39      if (error instanceof HttpErrorResponse && !authReq.url.includes('/api/login') && error.status === 401) {
40        return this.handle401Error(authReq, next);
41      }
42      if (error instanceof HttpErrorResponse && error.status === 403) {
43        return this.handle403Error();
44      }
45      return throwError(error);
46    }));
47  }

```

*Ci-dessus, la fonction intercept()*

Il permet d'intercepter une requête, si par exemple, le token d'un utilisateur est périmé, celui-ci recevra une réponse 401. Si l'url ciblée n'est pas "api/login", l'application va s'occuper de régénérer son token, de le re-login et d'ajouter le nouveau token dans les headers.

Si l'utilisateur n'a pas les permissions pour accéder à une ressource, l'application envoie une réponse 403 et le redirige sur l'accueil.

```

48  // tslint:disable-next-line:typedef
49  private handle401Error(request: HttpRequest<any>, next: HttpHandler) {
50    if (!this.isRefreshing) {
51      this.isRefreshing = true;
52      this.refreshTokenSubject.next( value: null );
53      const userLocalStorage = localStorage.getItem( key: 'currentUser' );
54      // @ts-ignore
55      const user = JSON.parse(userLocalStorage);
56      let refreshToken = null;
57      if (user) {
58        refreshToken = user.refreshToken;
59      }
60      if (refreshToken) {
61        return this.authService.loginUserByRefreshToken(refreshToken).pipe(
62          switchMap( project: (userToken: any) => {
63            this.isRefreshing = false;
64
65            return next.handle(this.addTokenHeader(request, userToken));
66          }),
67          catchError( selector: (err) => {
68            console.log('ERROR', err);
69            this.isRefreshing = false;
70            this.authService.logout();
71            return throwError(err);
72          })
73        );
74      }
75    }
76    return this.refreshTokenSubject.pipe(
77      filter( predicate: token => token !== null ),
78      take( count: 1 ),
79      switchMap( project: (token :B ) => next.handle(this.addTokenHeader(request, token)))
80    );
81  }
82  // tslint:disable-next-line:typedef
83  private handle403Error() {
84    // return this.authService.logout();
85    return this.router.navigate( commands: ['/recipes']);
86  }

```

*Ci-dessus, la gestion d'erreurs 401 et 403.*

```

88     private addTokenHeader(request: HttpRequest<any>, token: string) {
89
90         const userLocalStorage = localStorage.getItem( key: 'currentUser' );
91         // @ts-ignore
92         const user = JSON.parse(userLocalStorage);
93         if (user) {
94             token = user.token;
95         }
96
97         return request.clone( update: { headers: request.headers.set('Authorization', `Bearer ${token}`) } );
98     }
99

```

*Ci-dessus, la fonction qui permet d'ajouter le nouveau token dans les headers de la requête.*

Dans notre controller Back, nous n'avons qu'à vérifier l'utilisateur.

Pour ce faire, j'ai "extend" l' AbstractController de Symfony pour venir surcharger ma méthode `getUser()` et j'appelle cette dernière (`$this->getUser()`).

```

protected function getUser()
{
    $authorizationHeader = $this->request->headers->get( key: 'authorization' );

    if (! $authorizationHeader) {
        return null;
    }

    $authorizationHeaderArray = explode( separator: ' ', $authorizationHeader);

    if (
        count($authorizationHeaderArray) !== 2 ||
        ! (isset($authorizationHeaderArray[0]) && $authorizationHeaderArray[0] === 'Bearer')
    ) {
        throw new UnexpectedValueException( message: 'Header\'s structure is not compliant', code: 422);
    }

    $user = $this->userRepository->findOneBy([
        'token' => $authorizationHeaderArray[1],
    ]);

    if (! $user) {
        return null;
    }

    if ($user->getExpirationDateToken()->getTimestamp() < (new \DateTime( datetime: 'now'))->getTimestamp()) {
        return null;
    }

    return $user;
}

```

## Création d'une recette

J'ai commencé par rédiger ma méthode de création d'une nouvelle recette dans mon RecipeController.

J'ai ensuite créé le RecipeType, qui est le formulaire appelé dans la méthode de création.

```

69     /**
70      * @Route("/recipe", name="new_recipe", methods={"POST"})
71      */
72     public function new(Request $request): Response
73     {
74         $user = $this->getUser();
75         $data = json_decode($request->getContent(), associative: true);
76         if (!$data) {
77             return new JsonResponse(["message" => 'Bad JSON', 400]);
78         }
79         $recipe = new Recipe();
80         $form = $this->createForm( type: RecipeType::class, $recipe);
81         $form->submit($data);
82
83         if ($form->isSubmitted() && $form->isValid()) {
84             $updatedFile = $form->get('fileSource')->getData();
85
86             $this->uploadService->recipeThumbnail($updatedFile, $recipe);
87
88             $recipe
89                 ->setUser($user)
90                 ->setCreatedAt(new \DateTime( datetime: 'now' ));
91             $this->em->persist($recipe);
92             $this->em->flush();
93         } else {
94             $output = [];
95             return new JsonResponse($output, status: 400);
96         }
97         return $this->json([
98             'recipe' => $recipe->toArray(),
99             'message' => "success",
100            'statusCode' => 201,
101        ]);
102    }

```

*Ci-dessus, la méthode “new” dans RecipeController.*

```

25 J  ⌂ public function buildForm(FormBuilderInterface $builder, array $options)
26 {
27     $builder
28         ->add( child: 'title', type: TextType::class, [
29             'required' => true,
30             'constraints' => [
31                 new NotBlank([]),
32                 new Length([
33                     'min' => 3,
34                     'max' => 55,
35                 ]),
36             ],
37         ])
38         ->add( child: 'content', type: TextareaType::class, [
39             'required' => true,
40             'constraints' => [
41                 new NotBlank([]),
42                 new Length([
43                     'min' => 12,
44                     'max' => 4096,
45                 ]),
46             ],
47         ])
48         ->add( child: 'shortDescription', type: TextType::class, [
49             'constraints' => [
50                 new Length([
51                     'min' => 3,
52                     'max' => 280,
53                 ]),
54             ],
55         ])
56         ->add( child: 'file', type: FileType::class, [
57             'required' => false,
58             'data_class' => null,
59             'constraints' => [
60                 new Image([
61                     'maxSize' => '5M',
62                     'mimeTypes' => [
63                         'image/jpeg',
64                         'image/jpg',
65                         'image/png'
66                     ],
67                 ]),
68             ],
69         ])
70         ->add( child: 'fileSource')
71         ->add( child: 'ingredients', type: TextareaType::class, [
72             'required' => true,
73             'constraints' => [
74                 new NotBlank([]),
75                 new Length([
76                     'min' => 12,
77                     'max' => 4096,
78                 ]),
79             ],
80         ])
81         ->add( child: 'categories', type: EntityType::class, [
82             'class' => Category::class,
83             'by_reference' => false,
84             'choice_label' => 'name',
85             'multiple' => true,
86             'required' => true,
87         ])
88

```

```
56     |     =>add( child: 'difficulty', type: IntegerType::class, [
57     |     'required' => true,
58     |     'attr' => [
59     |         'pattern' => '/^[1-5]$/',
60     |     ],
61     |     'constraints' => [
62     |         new NotBlank([]),
63     |         new Positive([]),
64     |         new Length([
65     |             'max' => 1,
66     |         ]),
67     |         new Regex( pattern: '/^[1-5]$/'),
68     |     ],
69     | ),
70     | =>add( child: 'duration', type: IntegerType::class, [
71     |     'required' => true,
72     |     'constraints' => [
73     |         new NotBlank([]),
74     |         new Positive([]),
75     |         new Length([
76     |             'max' => 4,
77     |         ]),
78     |     ],
79     | ),
80     | =>add( child: 'personNb', type: IntegerType::class, [
81     |     'required' => true,
82     |     'constraints' => [
83     |         new NotBlank([]),
84     |         new Positive([]),
85     |         new Length([
86     |             'max' => 2,
87     |         ]),
88     |     ],
89     | ),
```

*Ci-dessus, le formulaire type RecipeType avec les contraintes de validation.*

```

10  class ImageUploadService
11  {
12      private $parameter;
13
14      public function __construct(ParameterBagInterface $parameter)
15      {
16          $this->parameter = $parameter;
17      }
18
19      public function categoryThumbnail($image, Category $category): void
20      {
21          $this->upload($image, $category);
22      }
23
24      public function recipeThumbnail($image, Recipe $recipe): void
25      {
26          $this->upload($image, $recipe);
27      }
28
29      public function userAvatar($image, User $user): void
30      {
31          $this->upload($image, $user);
32      }
33
34      public function upload($image, $entity): void
35      {
36          if ($image) {
37              $image_parts = explode( separator: ";base64,", $image);
38              $image_type_aux = explode( separator: "image/", $image_parts[0]);
39              $image_type = $image_type_aux[1];
40              $image_base64 = base64_decode($image_parts[1]);
41
42              $fileRelativePath = md5(uniqid()) . '.' . $image_type;
43              $fileAbsolutePath = $this->parameter->get('images_directory_absolute_path') . $fileRelativePath;
44              file_put_contents(($fileAbsolutePath), $image_base64);
45
46              $entity
47                  ->setFile($this->parameter->get('images_directory') . $fileRelativePath)
48                  ->setFileSource(null);
49          }
50      }
51
52  }

```

*Ci-dessus, le service appelé dans la méthode de création d'une recette pour ajouter une image. J'utilise ce service pour les images de recettes, de catégories et d'utilisateurs, ce qui me permet d'éviter de dupliquer du code.*

## Semi-croustillant

### Ajouter une recette

Titre :

Courte description :

Difficulté ( sur 5 ):

Durée ( en minutes ):

Nombre de personnes :

Image :

Catégories :

- Viandes
- Veggie
- Plats
- Entrée

Ingrédients :

- Choose heading ▾ | **B** *I* |
- Viande
  - Riz
  - Sel
  - Poivre

Etapes :

- Choose heading ▾ | **B** *I* |
- Cuire
  - Manger

## Upload d'image

Tout d'abord, je récupère mon image en “base64”. J’”explode” le tableau pour récupérer les données base64 et le type du fichier.

Je génère ensuite un nom de fichier en md5 que je concatène à l'extension du type de fichier ( png, jpg et cetera... ), j'enregistre ensuite le fichier dans ‘images\_directory’ ( que je définis au préalable dans services.yaml ).

Je “set” ensuite le chemin de l'avatar à mon entité car je ne souhaite pas enregistrer le fichier en base de données.

*Ci-dessous la méthode d'upload d'image dans mon service.*

```
public function upload($image, $entity): void
{
    if ($image) {
        $image_parts = explode( separator: ";base64,", $image);
        $image_type_aux = explode( separator: "image/", $image_parts[0]);
        $image_type = $image_type_aux[1];
        $image_base64 = base64_decode($image_parts[1]);

        $fileRelativePath = md5(uniqid()) . '.' . $image_type;
        $fileAbsolutePath = $this->parameter->get('images_directory_absolute_path') . $fileRelativePath;
        file_put_contents(( $fileAbsolutePath), $image_base64);

        $entity
            ->setFile($this->parameter->get('images_directory') . $fileRelativePath)
            ->setFileSource(null);
    }
}
```

## Compteur de vues

Pour créer mon compteur de vues sur les recettes, j'incrémente d'une unité chaque fois qu'un utilisateur consulte une recette et j'enregistre le compte en base de données.

```
103     public function show(Recipe $recipe): Response
104     {
105         $views = $recipe->getViewCount();
106         $recipe->setViewCount( viewCount: $views + 1);
107
108         $this->em->persist($recipe);
109         $this->em->flush();
```

*Cette partie de code se trouve dans la méthode “show” de mon RecipeController*

## Exporter une recette en PDF

Pour exporter une recette en PDF, j'utilise une fonction basique couplée à une media query.

*Ci-dessous, la fonction print() dans le fichier recipe-to-pdf.js*

```
43     ↗  print() {  
44       ↗  window.print();  
45     ↘ }  
      ↘ }
```

*Ci-dessous, la media query print qui me permet de donner un aspect spécifique à l'impression d'une recette ou l'enregistrement en PDF.*

```
92   ↗ @media print {  
93     ↗ .recipe-header {  
94       ↗ .img {  
95         ↗ display: none;  
96       ↘ }  
97     ↘ }  
98     ↗ .recipe-title{  
99       ↗ h2 {  
100        ↗ color: $sc-black;  
101        ↗ background-color: white;  
102        ↗ backdrop-filter: none;  
103      ↘ }  
104    ↘ }  
105    ↘ }  
106    ↘ }  
107  ↘ }  
108  ↘ }  
109  ↘ }  
110  ↗ .recipe-top {  
111    ↗ display: none;  
112  ↘ }  
113  ↘ }  
114  ↗ .recipe {  
115    ↗ .published-at {  
116      ↗ display: none;  
117    ↘ }  
118    ↗ .difficulty {  
119      ↗ padding-top: 2rem;  
120    ↘ }  
121    ↗ .btn-back {  
122      ↗ display: none;  
123    ↘ }  
124  ↘ }  
125  ↘ }  
126  ↘ }  
127  ↘ }  
128  ↘ }
```

## Sécurité

### isAdmin

Cette méthode me permet de vérifier qu'un utilisateur ait les droits pour accéder aux fonctionnalités d'administrateur ( Crud Catégories ).

```
public function getIsAdmin(): ?bool
{
    return $this->isAdmin;
}
```

Cela me permet de contrôler l'accès directement dans le controller, par exemple, je vérifie que l'utilisateur courant est bien en `isAdmin() === true` afin qu'un utilisateur enregistré ne puisse pas créer de nouvelle catégorie.

```
63     /**
64      * @Route("/admin-categories", name="admin_categories", methods={"GET"})
65
66      */
67     public function indexAdmin(): Response
68     {
69         $user = $this->getUser();
70
71         if ($user->getIsAdmin() === false) {
72             return new JsonResponse(["message" => 'You are not allowed to do this'], status: 403);
73         }
74     }
```

*Ci-dessus, le CategoryController, accessible uniquement par l'admin.*

## Side nav

Je contrôle aussi dans le front. Par exemple, pour ma “sidenav” je vérifie si l’utilisateur courant est admin pour lui afficher la gestion des catégories.

```
<div class="admin-bo" *ngIf="user.isAdmin === true">
    <span class="admin-label">Paramètres d'administrateur</span>
    <ul>
        <li>
            <a [routerLink]=['/admin-categories']>
                data-bs-toggle="collapse"
                data-bs-target="#sideNavBtn">
                    <span class="me-3">
                        <fa-icon class="cog-icon" [icon]="faCog"></fa-icon>
                    </span>
                    <span class="side-nav-label">Gestion des catégories</span>
            </a>
        </li>
    </ul>
</div>
/div>
```

## DQL

Doctrine est l’ORM utilisé par Symfony. Il permet de rajouter une couche de protection lorsque l’on souhaite accéder / persister / supprimer des données. Doctrine Query Language utilise les requêtes préparées ( la requête est pré-interprétée et peut recevoir des paramètres ).

## Validation constraints ( FormType )

Les contraintes de validation permettent de contrôler les données saisies et soumises par un utilisateur dans un formulaire.

```
56     ->add( child: 'difficulty', type: IntegerType::class, [
57         'required' => true,
58         'attr' => [
59             'pattern' => '/^1-5$/',
60         ],
61         'constraints' => [
62             new NotBlank([]),
63             new Positive([]),
64             new Length([
65                 'max' => 1,
66             ]),
67             new Regex(pattern: '/^1-5$/'),
68         ],
69     ],
70     ->add( child: 'duration', type: IntegerType::class, [
71         'required' => true,
72         'constraints' => [
73             new NotBlank([]),
74             new Positive([]),
75             new Length([
76                 'max' => 4,
77             ]),
78         ],
79     ],
80     ->add( child: 'personNb', type: IntegerType::class, [
81         'required' => true,
82         'constraints' => [
83             new NotBlank([]),
84             new Positive([]),
85             new Length([
86                 'max' => 2,
87             ]),
88         ],
89     ],

```

*Ci-dessus, les contraintes de validation sur le formulaire des recettes.  
On peut voir par exemple une Regex sur le champ “difficulté”.*

## Vérifications côté Angular

Je vérifie aussi côté client les champs des formulaires dans le fichier .ts et .html de mon “component”.

```

30 myForm = new FormGroup( controls: {
31   title: new FormControl( formState: '', validatorOrOpts: [Validators.required, Validators.minLength( minLength: 3)]),
32   content: new FormControl( formState: '', validatorOrOpts: [Validators.required]),
33   shortDescription: new FormControl( formState: '', validatorOrOpts: [Validators.required]),
34   difficulty: new FormControl( formState: 0, validatorOrOpts: [Validators.required, Validators.min( min: 1), Validators.max( max: 5)]),
35   duration: new FormControl( formState: 0, validatorOrOpts: [Validators.required, Validators.min( min: 1), Validators.max( max: 9999)]),
36   personNb: new FormControl( formState: 0, validatorOrOpts: [Validators.required, Validators.min( min: 1), Validators.max( max: 99)]),
37   file: new FormControl( formState: '', validatorOrOpts: [Validators.required]),
38   fileSource: new FormControl( formState: '', validatorOrOpts: [Validators.required]),
39   categories: new FormControl( formState: [], validatorOrOpts: [Validators.required]),
40   ingredients: new FormControl( formState: [], validatorOrOpts: [Validators.required])
41 });

```

*Ci-dessus, les vérifications dans recipe-add.component.ts.*

```

47 <label for="guests">Nombre de personnes :</label>
48 <input formControlName="personNb"
49   id="guests"
50   class="form-control"
51   placeholder="nb de personnes"
52   type="number"
53   min="1"
54   max="99"
55   required="required"/>

```

*Ci-dessus, les vérifications dans recipe-add.component.html.*

## CI / CD - TESTS

Etant donné que je stocke mon projet sur Gitlab, j'utilise gitlab-ci et j'intègre mes tests.

( Les pipelines sont en “manual” car gitlab impose des limitations sur les comptes gratuits, ainsi les pipelines ne se lancent pas à chaque “push” )

```

1  stages:
2    - tests
3    - build
4
5  tests:
6    stage: tests
7    image: php:7.4-cli
8    script:
9      - cd dev/api
10     - apt-get update
11     - apt-get install zip unzip
12     - php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
13     - php composer-setup.php
14     - php -r "unlink('composer-setup.php');"
15     - php composer.phar install --no-scripts
16     - cp .env.test .env
17     - php bin/phpunit
18   rules:
19     - when: manual
20
21 build docker image:
22   stage: build
23   image: docker
24   services:
25     - docker:dind
26   script:
27     - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
28     - docker build -t registry.gitlab.com/jordybasteid/SC_v2/api:latest dev/api
29     - docker push registry.gitlab.com/jordybasteid/SC_v2/api:latest
30     - docker build -t registry.gitlab.com/jordybasteid/SC_v2/app:latest dev/app
31     - docker push registry.gitlab.com/jordybasteid/SC_v2/app:latest
32   rules:
33     - when: manual

```

*Ci-dessus, le fichier gitlab-ci.yml*

J'ai réalisé deux simples tests unitaires pour le moment. Je teste une méthode magique de PHP `__toString()` et une méthode `build()` de ma classe `Recipe`.

```

324     public function __toString()
325     {
326         return $this->title;
327     }
328
329     public static function build(string $title, string $content): Recipe
330     {
331         return $recipe = (new Recipe())
332             ->setTitle($title)
333             ->setContent($content);
334     }

```

*Ci-dessus, les méthodes à tester de ma classe Recipe.*

```

8 ► class RecipeTest extends TestCase
9
10 ►     public function testToString(): void
11     {
12         $recipe = new Recipe();
13         $recipe->setTitle( title: 'Test title');
14         $check = 'Test title';
15
16         $assign = $recipe->__toString();
17
18         $this->assertEquals($check, $assign);
19     }
20
21 ►     public function testBuild(): void
22     {
23         $title = 'test title';
24         $content = 'test content';
25
26         $result = Recipe::build($title, $content);
27
28         $this->assertEquals($title, $result->getTitle());
29         $this->assertEquals($content, $result->getContent());
30     }
31

```

*Ci-dessus, mon fichier de test.*

Lorsque l'on regarde l'état de la pipeline sur gitlab, on peut voir à la fin de la stage "tests" que nos tests se sont bien déroulés.

```
293 $ php bin/phpunit
294 PHPUnit 9.5.21 #StandWithUkraine
295 Testing
296 ..
297 Time: 00:00.016, Memory: 10.00 MB
298 OK (2 tests, 3 assertions)
✓ 300 Cleaning up project directory and file based variables
302 Job succeeded
```

## CONCLUSIONS

Cette version du projet est fonctionnelle en l'état.

Cependant je pensais avoir le temps d'aller plus loin et de rajouter quelques fonctionnalités comme un système de notifications, une page de contact, de mentions légales et cetera...

Ces fonctionnalités seront donc mises en place dans les futures versions de l'application (*voir page 12*).

Je me suis rendu compte à mi parcours que la charge de travail restante était assez conséquente car je souhaitais réaliser une application la plus réaliste possible.

Ce projet m'a permis d'apprendre beaucoup de choses et de monter en compétences.

Il me tenait à cœur de réaliser un projet orienté mobile et donc avec des fonctionnalités que l'on retrouve sur la plupart des gros sites web actuels ( scroll infini, OAuth et cetera... )

Si je devais recommencer ce projet je pense que j'essaierai de bien séparer mon API et mon front en ayant deux repositories différents. Je choisirai aussi sans doute un autre framework front que Angular.

Merci pour votre attention.