

# Dossier de projet

Sicaire Loïc – Bachelor 3 EPSI



# Table des matières

ABSTRACT .....	4
INTRODUCTION .....	5
CONTEXTE PROFESSIONNEL .....	6
Environnement professionnel .....	6
Environnement technique .....	7
I – PRESENTATION DU PROJET .....	9
Le projet ConnectMyResto .....	9
Cahier des charges .....	9
III – CREATION DU WEBSERVICE .....	11
Configuration .....	11
Développement .....	13
Production.....	16
IV – CREATION D’UNE INTERFACE GRAPHIQUE DE GESTION .....	17
Installations pré requises .....	17
Suivi des utilisateurs .....	18
Module de gestion des informations d’un restaurant.....	20
Module de gestion des horaires .....	22
Module de gestion de la carte .....	24
Module de gestion des zones de livraison.....	26
Module de gestion des moyens de paiement .....	27
Module de gestion des partenariats.....	28
Module de gestion des commandes.....	29
V – CONCEPTION D’UN BACKOFFICE POUR LES APPLICATIONS .....	31
Module de gestion des contrats .....	32
Module de gestion des restaurants partenaires .....	33
Module de gestion du compte.....	34
Module de visualisation d’un restaurant.....	36
VI – AMELIORATIONS POSSIBLES .....	38
WebService .....	38
Interface de gestion restaurateurs .....	38
Back-office marketplaces.....	39
Infrastructure globale .....	39
VII – CONCLUSION.....	40

ANNEXE 1 – LISTE DES COMPETENCES COUVERTES PAR LE PROJET .....	41
ANNEXE 2 – ORGANIGRAMME DE LA SOCIETE .....	42
ANNEXE 3 – CLASSE AUTHTokenAUTHENTICATOR .....	43
ANNEXE 4 – CREATION D'UNE COMMANDE .....	44
ANNEXE 5 – AJOUT / SUPPRESSION D'UN PRODUIT .....	45
ANNEXE 6 – CONFIRMATION D'UNE COMMANDE .....	46
ANNEXE 7 – POSTMAN PAR GOOGLE / ELABORATION D'UN JEU DE REQUETES .....	47
ANNEXE 8 – DOCUMENTATION DE L'API .....	48

## Abstract

During this year, I worked in Mate-maker company, as a Full-stack developer in the technical service. I had so much freedom for the realisations i had to do. I had to be really autonomous because i was the only one that had some real technical skills from my education at EPSI school.

The project i had to realize was an ambitious one that requires me to do my best and develop my abilities in web development. Indeed, the tasks i had to perform concerned the food technology, a world that i didn't know before. Some food applications that allows their customers to send orders to restaurants and been delivered at home already exists, but these applications are a lot and the restaurants can't work with each of them. The goal of this project, that we called ConnectMyResto was to make life easier for restaurant owners, without sacrificing applications.

To do this, I thought about creating a solution that centralizes all of these applications in the same endpoint, and allow them to see restaurant's informations and send them some orders.

To complete this solution, i had to do another platform for the restaurants, so that they may update their informations and see the orders that the marketplaces (applications) send to them.

Another part of my job was to design a backoffice for these marketplaces to check their relations with our partners and update their account and cash-in some money.

These are all the tasks that I will present along this project file.

## Introduction

Dans le cadre de l'obtention de ma 3<sup>ème</sup> année d'études en informatique, j'ai réalisé de nombreuses missions au sein de la société Mate-maker, en tant qu'alternant. Mate-maker c'est une start-up qui a pour projet de connecter très simplement les restaurateurs avec les sociétés de livraison / référencement (type AlloResto/UberEats...), de manière centralisée.

Au cours de cette année j'ai pu fortement développer mes compétences, non seulement en informatique mais aussi dans des domaines transversaux tels que l'économie, le marketing, l'entrepreneuriat...

Disposant d'une autonomie quasi-totale, j'ai eu à réaliser :

- Un Webservice HTTP (API) permettant aux applications de commande / livraison / référencement de restaurant d'utiliser les données fournies le plus simplement possible (et ainsi accroître leur choix de restaurants), ainsi qu'envoyer des commandes aux restaurateurs.
- Une documentation interactive de l'API accessible à tous
- Une interface de gestion pour les restaurateurs leur fournissant tous les outils pour gérer les informations de son restaurant (afin de les transmettre ensuite aux applications connectées en temps réel), et suivre les commandes qui lui sont envoyées par ces applications.

Afin de mener ces missions à bien, il m'a fallu dans un premier temps et avec l'aide de mon tuteur, cerner les besoins des utilisateurs de notre produit, puis construire une structure de données adéquate.

Une fois la base de données créée, j'ai pu travailler sur la conception de l'API, de manière à rendre les ressources transmises les plus simples possibles.

Dans un second temps, j'ai eu à m'atteler à la création de l'interface dédiée aux restaurateurs, dans une première version sous Android en Java pour Android, puis dans un second temps sous la forme d'une interface web en utilisant le Framework Symfony et son architecture MVC. Enfin

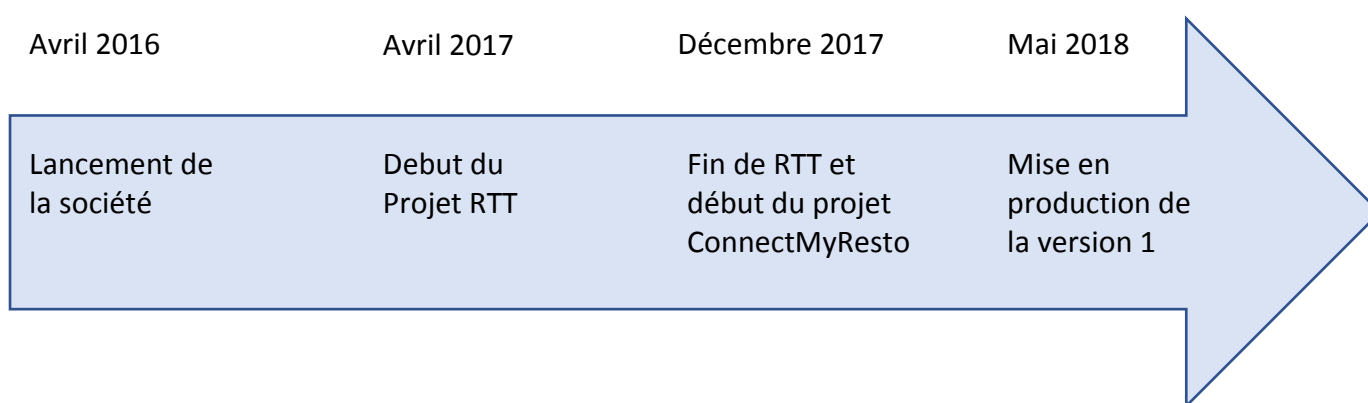
Enfin, j'ai dû concevoir un backoffice pour les utilisateurs de l'API, de manière à leur permettre de gérer les informations de leurs comptes ainsi que leurs relations avec les restaurants.

## Contexte professionnel

### Environnement professionnel

#### L'histoire de Mate-maker

La société Mate-maker est une petite SAS (société par action simplifiée) fondée en 2016, travaillant au cœur de l'innovation, et comptant quatre associés. J'étais quant à moi alternant au sein du service technique (*cf. Annexe 1 - organigramme de la société*). Parmi les salariés/associés nous étions deux à œuvrer pour mener à bien ce projet, les autres associés ne travaillant pas directement au sein de l'entreprise.



#### Les valeurs de l'entreprise

Je me suis intégré assez facilement au sein de la société car mon caractère correspond parfaitement aux principes de cette dernière. Nous avançons dans un cadre propice au développement personnel, et il est important pour chacun de partir en ayant obtenu des compétences supplémentaires par rapport à celles acquises avant l'arrivée dans la boîte. Si je devais résumer les objectifs de la société en général, je dirais que nous sommes à la recherche constante du progrès, c'est à dire qu'au-delà de son travail dans la startup, chaque collaborateur doit avoir à sa disposition les moyens d'avancer sur le plan individuel.

J'ai moi-même acquis de nombreuses compétences, par l'autonomie que la startup m'a laissée, en me renseignant sur des tâches sur lesquelles je n'avais pas encore reçu de formation (développement d'API, maîtrise de Symfony, utilisation d'autres API...)

#### Mon rôle au sein de l'entreprise

Durant cette année, j'ai pu mettre à profit librement les connaissances que j'ai acquise au cours de ma formation dans une quasi-totale autonomie, concernant toute la partie technique du projet. En effet, du choix des technologies au lancement de l'application, en passant par des phases de maquettage, de configurations..., je me suis montré force de proposition et ait reçu une très grande confiance puisque la plupart de ces propositions ont été adoptées pour la suite.

## Environnement technique

### Dispositifs matériels

Afin de mener à bien le projet, j'avais à ma disposition dans les locaux un ensemble d'outils. Pour le développement de la solution, nous avons 2 ordinateurs munis d'un système d'exploitation Windows 10 ainsi qu'une salle de détente dans laquelle nous avons plaisir à débattre sur les futurs choix primordiaux de l'application. C'est aussi dans cette salle que se déroulaient les tests grandeur nature. J'avais également à ma disposition un contrat d'hébergement 1&1 (formule Basic) associé à plusieurs noms de domaine, ainsi que 2 base de données MySQL.

### Dispositifs logiciels

Au niveau logiciel, nous avons utilisé des langages et technologies web principalement au vu de la souplesse qu'elles apportent. Mon choix en matière de technologie était basé sur 2 critères essentiellement : ma maîtrise de ces dernières et les avantages qu'elles apportent.

En effet, il fallait choisir des outils sur lesquels j'avais reçu un minimum de formation afin mener le plus rapidement possible le projet à bien. Ainsi, j'ai choisi de me servir du langage php ainsi que du framework Symfony (version 3.4) et de son architecture MVC pour sa fiabilité et sa maintenabilité. Je me suis aussi servi de 2 bundles issus de friendsofsymfony : FOSRestBundle permettant de construire aisément des API REST sous symfony ainsi que FOSUserBundle permettant la gestion des utilisateurs (avec restriction des accès, espace membre...). Enfin pour l'affichage des données coté client, j'ai utilisé des langages de mise en page tels que HTML / CSS et javascript, équipé de la bibliothèque JQuery permettant ainsi de donner un coté dynamique et simple à l'application.



## Maquettage

J'ai dû présenter aux restaurateurs une maquette de la première version de l'application pour les restaurants sous Android. Pour cela, j'ai utilisé les outils suivants :

- Android Studio
- Le langage Java et XML pour la mise en place de l'application
- Le Android SDK version 21
- Des plans de l'application sur papier.

## Aides externes / API utilisées

J'ai utilisé au cours du développement de ce projet plusieurs technologies externes provenant de sources variées :

- J'ai utilisé l'API de la société Stripe afin de gérer les paiements en ligne simplement, et de proposer une gestion de soldes à moindre coûts.
- Je me suis servi des API Google Maps et Google Places pour la géolocalisation des restaurants et le calcul de distances entre un restaurant et un client.
- J'ai aussi pu utiliser l'API mailjet à des fins commerciales pour assurer un suivi total des mails envoyés aux restaurants / applications susceptibles de rejoindre le projet



# I – Présentation du projet

## Le projet ConnectMyResto

### Clarification du problème

Avec l'évolution de la « food-technology », les restaurateurs sont de plus en plus nombreux à vouloir rendre disponible leur carte en ligne afin d'augmenter leur nombre de commandes et ainsi leur chiffre d'affaires. Pour cela, beaucoup font appels à des sociétés tierces (comme Deliveroo, JusEat...). Cependant ces derniers proposent chacun leur propre système et les restaurants sont donc obligés de s'adapter à chacun d'eux. Ainsi être présent sur plusieurs plateformes différentes est très compliqué à gérer pour les restaurant (accumulation de tablettes, sites web...).

### Solution apportée

Pour résoudre ce problème, nous avons pensé que la meilleure solution était de proposer un système unique avec lesquels les sociétés de livraison/commandes pourraient interagir et envoyer des commandes. Grâce à ce système, les restaurateurs n'utiliseraient qu'une seule et même interface pour configurer les informations de leur restaurant, lesquelles seraient envoyées aux apporteurs d'affaires qui auraient ainsi accès à des données en temps réel. Nous avons nommé cette solution « ConnectMyResto ». Cette solution permettrait ainsi de créer une norme pour le passage de commandes voire pour la collaboration en général avec les restaurants.

## Cahier des charges

### Définition des besoins

Pour la création d'un système universel comme celui de ConnectMyResto, de nombreux critères sont à prendre en compte techniquement parlant.

- Il s'agissait de créer une interface de gestion très simple et intuitive pour les restaurateurs, pas forcément habitués à travailler avec de tels outils. Ces derniers devraient disposer d'une interface permettant la gestion de leurs informations (coordonnées postales, gestion des produits et leurs stock, zones de livraisons pour ceux possédant un service de livraison, horaires & agenda, contrats avec les différents apporteurs d'affaires, récapitulatif et gestion des commandes).
- En ce qui concerne les sociétés tierces dites « apporteurs d'affaires », elles devaient disposer d'une interface leur permettant de recueillir ces informations pour les afficher à leurs clients, en temps réel et d'envoyer des commandes aux restaurateurs. De plus il fallait leur proposer une solution leur permettant de sélectionner facilement des restaurants afin de leur soumettre des contrats, et assurer le suivi de ces contrats pour les deux parties.

Pour créer un tel service, il a fallu donc conjuguer entre la facilité d'implémentation pour que le marché l'adopte facilement et la "complétude" afin de répondre à un maximum de cas particuliers.

## Analyses du système

Les besoins établis, une analyse claire de ces derniers était nécessaire afin de bien les comprendre et ne pas se tromper sur les attentes des futurs utilisateurs, et ainsi réaliser une solution capable de prendre en compte toutes les spécificités. Au vu de la complexité évidente du système global, prenant en compte beaucoup de points, j'ai effectué une première analyse merisienne de ce dernier en le divisant en sous-systèmes plus faciles à étudier. Sont donc sortis de cette analyse plusieurs Modèles Conceptuels des Données (MCD) représentant chacun un de ces sous-systèmes.

Pour pouvoir proposer un tel service, il m'a paru immédiatement évident que contruire une interface de programmation applicative (Application Programming Interface ou API) allait être le moyen le plus efficace pour que les apporteurs d'affaires puissent utiliser aisément nos services. En effet, grâce à ce type de solution, les applications authentifiées souhaitant récupérer les restaurants ou leur passer des commandes peuvent interroger notre serveur qui leur fournira le(s) service(s) souhaité(s).

La conception de la base de données ayant été réalisé via Doctrine, j'ai laissé le soin à l'ORM de générer pour moi les tables dont j'ai eu besoin pour fabriquer une structure de données adaptées à mon modèle. (En prenant bien-sur toujours le soin de vérifier les requêtes que ce dernier allait executer).

## III – Création du Webservice

### Configuration

#### Installation de Symfony – Création d'entités

La version de Symfony que j'ai choisie était la version 3.4, possédant une documentation encore maintenue et pour laquelle j'ai reçu une formation. Symfony est équipé de l'ORM Doctrine, permettant de gérer plus simplement le lien entre la base de données et les classes PHP créées. On ne parle alors plus de classe mais bien d'entités. Ces entités sont gérées par doctrine par le biais d'un EntityMangager, service permettant d'effectuer simplement des actions en temps réel sur vos entités (et ainsi sur vos données). Un tel outil est très pratique lorsqu'il s'agit de bâtir des applications en lien très proche avec une base de données, cependant il convient de ne pas faire une confiance aveugle à un ORM, et de toujours vérifier les actions effectuées lors de la phase de développement. C'est pourquoi j'inspectais régulièrement les logs et prenais la précaution d'inspecter systématiquement les requêtes SQL avant qu'elles ne soient exécutées sur la base, lors de la modification d'entités, grâce à la commande `php bin/console doctrine:schema:update --dump-sql`.

#### Installation / configuration de FOSRestBundle

Afin de simplifier le code d'accès aux ressources, friendsofsymfony propose un bundle conçu spécialement pour la gestion d'API REST sous Symfony : FOSRestBundle. Ce bundle apporte des facilités pour le code, notamment grâce à des annotations permettant de décrire la route et la méthode HTTP qui déclenchera une méthode PHP chargée de gérer une ressource, ainsi que les groupes de sérialisation associés. J'ai donc configuré le bundle dans le config.yml afin qu'il utilise le sérialisateur natif de Symfony, de manière à renvoyer les données dans un format hautement universel et très économe en ressources : le JSON.

Extraits de la configuration de Symfony et de FOSRestBundle (fichier `app/config/config.yml`)

```
fos_rest:
  exception:
    enabled: true
    messages:
      'Symfony\Component\Security\Core\Exception\BadCredentialsException': true
      '\Symfony\Component\HttpKernel\Exception\BadRequestHttpException': true
    codes:
      'Symfony\Component\Security\Core\Exception\BadCredentialsException': 401
  routing_loader:
    include_format: false
  view:
    view_response_listener: true
  format_listener:
    rules:
      - { path: '^/', priorities: ['json'], fallback_format: 'json' }
```

### Extrait de la configuration de Symfony

```
serializer:  
  enabled: true
```

## Système d'authentification

Afin d'authentifier les utilisateurs de l'API, et ainsi avoir un contrôle des accès efficaces, j'ai décidé d'implémenter la SimplePreAuthenticatorInterface ([voir annexe 3 – classe AuthTokenAuthenticator](#)) nativement installé dans Symfony pour faire en sorte que les utilisateurs possèdent chacun une clé d'API unique, qu'ils doivent envoyer pour chacune des futures requêtes qu'ils feront. Les requêtes envoyées avec une clé incorrecte ou sans clé se voient ainsi rejetées avec pour réponse une erreur de type 401 UNAUTHORIZED. Il reste à déclarer via le firewall de Symfony (configurable dans le security.yml) que les requêtes passent systématiquement par le provider ainsi créé.

### Extrait de la configuration du pare-feu Symfony pour l'authentification des requêtes (fichier app/config/security.yml)

```
firewalls:  
  # désactive l'authentification pour l'accès aux assets et au profiler  
  dev:  
    pattern: ^/(_(profiler|wdt)|css|images|js)/  
    security: false  
  
  main:  
    pattern: ^/  
    stateless: true  
    simple_preauth:  
      authenticator: auth_token_authenticator  
    provider: auth_token_user_provider  
    anonymous: ~
```

La création d'utilisateur est soumise elle-même à une authentification, ce qui implique que seul un utilisateur peut créer un autre utilisateur, afin de contrôler la création de comptes sur notre système.

## Développement

### Génération des contrôleurs

Comme expliqué précédemment, FOSRestBundle nous facilite le code de l'API en s'occupant automatiquement de la négociation de contenu, c'est-à-dire détection des méthodes/routes/contenus/entêtes de la requête HTTP... J'ai pu ainsi facilement, par le biais d'annotations utilisées dans des contrôleurs générés grâce à la commande *generate:controller*, gérer les requêtes envoyées par le client. Ces annotations permettent aux méthodes PHP situées dans les contrôleurs d'être appelées automatiquement lorsqu'une requête est envoyée avec une certaine méthode HTTP sur une certaine route. Ces requêtes seront donc traitées automatiquement par une méthode d'un contrôleur grâce à l'annotation `@Rest\<methode>(<url>)`.

Ainsi, une requête GET sur l'URL `/restaurants` par exemple se verra être traitée par la première méthode qui possèdera l'annotation `@Rest\Get(« /restaurants »)`.

#### *Inclusion des annotations @Rest du bundle FOSRestBundle*

```
use FOS\RestBundle\Controller\Annotations as Rest;
```

#### *Utilisation des annotation @Rest\View() et @Rest\<method>(), avec emploi d'un groupe de sérialisation*

```
/**
 * @Rest\View(serializerGroups={"restaurant"})
 * @Rest\Get("/restaurants/{id}")
 */
public function getRestaurantAction(Request $request) {...}
```

### Création des requêtes

Ensuite, j'ai eu à me confronter au cœur du sujet : comment les apporteurs d'affaire souhaiteront interagir avec notre système. Il m'a fallu réfléchir et proposer un jeu de requêtes le plus simple possible à comprendre (afin qu'elles soient faciles à prendre en main) renvoyant les informations strictement nécessaires au bon fonctionnement des applications susceptible d'être intéressés par notre API.

Le tableau ci-dessous représente la liste des principales requêtes que j'ai mis en place pour permettre aux apporteurs d'affaire de récupérer les informations des restaurants partenaires et de leur passer des commandes.

Methode HTTP	Route	Paramètres URL	Paramètre corps de la req.	Description
GET	/restaurants			Renvoie la liste de tous les restaurants
GET	/restaurants/{ <i>id</i> }	<i>Id</i> : identifiant du restaurant		Renvoie un restaurant grâce à son identifiant
GET	/restaurants/delivery/{ <i>code</i> }	<i>Code</i> : code INSEE de la commune de livraison		Renvoie les restaurants livrant une commune
GET	/restaurants/country/{ <i>code</i> }	<i>Code</i> : code du pays de la recherche		Récupère le nombre de restaurants sur tout un pays
POST	Restaurants/{ <i>id</i> }/create-order	<i>Id</i> : identifiant du restaurant	<i>orderType</i> : identifiant du type de commande + Informations de la commande	Crée une commande vide avec les informations passées en paramètre
POST	/orders/{ <i>id</i> }/update	<i>Id</i> : Identifiant de la commande pré-crée	+ Informations de la commande	Modifie les informations de base d'une commande
POST	/orders/{ <i>id</i> }/add-product	<i>Id</i> : Identifiant de la commande pré-crée	<i>Id</i> : identifiant du produit à ajouter <i>Options</i> : tableau des propriétés du produit <i>Supplements</i> : Tableau des suppléments du produit	Ajoute un produit à une commande existante et met à jour ses info.
DELETE	/orders/{ <i>id</i> }/delete-product/{ <i>referenceKey</i> }	<i>Id</i> : Identifiant de la commande pré-crée <i>referenceKey</i> : référence unique du produit de la commande		Supprime un produit d'une commande
POST	/orders/{ <i>id</i> }/confirm	<i>Id</i> : Identifiant de la commande pré-crée		Confirme une commande si ses informations sont correctes

## Explicitation du système de commandes

Le système de commande était assez complexe et délicat à mettre en place au vu de son aspect financier. En effet, j'ai été confronté à plusieurs problèmes dont le premier : « comment garantir aux restaurants l'intégrité et la fiabilité des commandes envoyées par les applications ? ». C'est pour cela que j'ai mis en place la première mesure : restreindre l'accès à l'API (surtout à la création de commande) aux applications autorisées, et vérifiées au préalable. J'ai donc premièrement cru que cette sécurité suffirait et ait créé une simple requête pour créer une commande, mettant toutes les informations à jour d'un seul coup. Cependant, cette solution est assez binaire : L'application cliente ne pouvait savoir s'il avait la possibilité de créer une commande sur ce restaurant avec tels produits que lors du passage de la requête finale, ce qui peut nuire à son expérience utilisateur. C'est pourquoi j'ai opté pour une solution en 3 temps pour le passage d'une commande :

- **Création de la commande / modification** : remplissage des informations nécessaires à la commande (le type de commande : en livraison, à emporter...). *Voir annexe 4 – Création d'une commande*
- **Ajout/suppression des produits** : pour avoir des informations en temps réel sur la disponibilité d'un produit, ils doivent s'ajouter au fur et à mesure de la commande. Chaque produit associé à une commande possède une clé unique pour pouvoir le supprimer. *Voir annexe 5 – Ajout de produit*
- **Confirmation de la commande** : une fois la commande remplie et dans un état « Confirmable », le client peut confirmer sa commande, qui sera ainsi envoyée telle qu'elle au restaurant concerné avec un statut « Commandé ». Une fois la commande dans cet état, il n'est plus possible pour le client de la modifier. *Voir annexe 6 – Confirmation d'une commande*

Bien entendu, seul le créateur d'une commande (authentifié par sa clé d'API) peut effectuer ces actions sur une commande déjà créée,

Au fur et à mesure que la commande se crée et se remplit, j'ai conçu un système d'erreurs qui permettent aux clients d'être informés sur les informations manquantes et/ou erronées dans la commande afin que cette dernière passe dans un état « Confirmable »

## Explicitation du système de restaurant

Le système de restaurants m'a fait me pencher sur les réelles attentes des futurs utilisateurs de l'API.

Pour ce faire, j'ai décidé d'analyser les différentes API de restaurants disponibles gratuitement en ligne (...) ainsi que les principales plateformes permettant la commande/réservation/retrait en ligne (JustEat, UberEats, Foodora...) et de m'en inspirer pour renvoyer les données les plus cohérentes possibles.

Pour des soucis de confidentialité, j'ai restreint l'accès à la carte d'un restaurant aux utilisateurs ayant un contrat en cours et accepté par ce même restaurant. Afin d'économiser du temps de traitement dans les requêtes de recherche, la carte n'est disponible que lors de la requête permettant d'afficher 1 restaurant.

Il fallait donc proposer à ces utilisateurs un système simple et efficace, qui leur permettrait d'obtenir les informations souhaitées en un rien de temps. C'est pourquoi j'ai élaboré des requêtes de recherche de restaurants permettant de filtrer les données ainsi renvoyées grâce à des paramètres d'URL facultatifs. Toutes les requêtes renvoyant + d'un restaurant acceptent ces paramètres. Ces paramètres permettent de renvoyer les résultats souhaités filtrés par des critères que précise le tableau suivant :

Nom du paramètre	Valeurs possibles	Description
Partnership	None, waiting, accepted, refused, all	Ressort uniquement les résultats ayant le statut de partenariat passé en paramètre : pas de partenariat, en attente, accepté, refusé
PaymentMode	CB, CB_ELEC, CV, CD...	Renvoie uniquement les résultats acceptant les moyens de paiement passés en paramètre. Ces noms sont amenés à être changé au fur et à mesure de notre avancée, toute modification sera disponible sur la documentation officielle.
Dates	Suite de dates (au format YYYY-mm-dd), séparées par des virgules	Permet à l'utilisateur de récupérer les horaires du restaurant pour les dates précisées. Par défaut les dates sont aujourd'hui et demain.

## Production

### Tests

Afin de tester le bon fonctionnement de l'API avant la mise en production finale, je disposais d'un environnement similaire à celui de production ainsi que d'une base de données de test pour ne pas compromettre les données du serveur de production. Ainsi, le code source était d'abord déployé sur le serveur de test pour vérifier chacune des requêtes (préalablement vérifiées en phase de développement bien sûr). Pour faciliter les tests, j'ai créé un jeu de requêtes à l'aide du logiciel Postman de Google, et ait automatisé le lancement de toutes les requêtes disponibles sur l'API, pour chacun des serveurs (développement, test, production). ***Voir annexe 7 – Postman par Google.***

### Versionning / Maintenance

Pour suivre et faire évoluer le service que je venais de créer, j'ai mis en place un logiciel de gestion de version : l'outil de versionning Git. Avec ce logiciel, on peut suivre l'évolution du code et ainsi revenir à des versions précédentes si on le souhaite, simplement en créant un dépôt avec le commande *git init*. Avec git, j'ai utilisé le dépôt distant GitHub, afin de partager le code source avec mon tuteur en entreprise.

### Documentation

La qualité d'une API comme celle-ci se définit à 50% par son code (sa maintenabilité, sa cohérence des informations renvoyées, sa simplicité d'utilisation...) et à 50% par sa documentation (bonne compréhension du système, de l'utilisation...). C'est pourquoi j'ai opté pour l'élaboration d'une documentation concise mais claire par le biais d'une interface web. En effet, il me paraissait beaucoup plus efficace de proposer une plateforme consacrée à l'explication des différentes utilisations possibles de la solution. J'ai donc créé cette interface web à l'aide des langages HTML/CSS/Javascript, ainsi que des librairies JQuery, highlight.js et spin.js, afin de proposer une documentation lisible aisément et de rendre la navigation beaucoup plus attrayante. ***Voir annexe 8 – Documentation de l'API***



## IV – Création d’une interface graphique de gestion

### Installations pré requises

#### Choix des technologies

L’interface devait être à la disposition de n’importe quel restaurateur, le plus facilement possible et d’une manière peu coûteuse. C’est pourquoi, après avoir constaté que la plupart des restaurateurs d’aujourd’hui possédaient des tablettes android pour utiliser les applications des apporteurs d’affaire, j’ai au départ choisi la technologie ANDROID et décidé de réaliser cette plateforme dans le langage natif d’android : Java. Cependant, après l’élaboration d’une première version (peu sophistiquée, mais fonctionnelle) , nous sommes allés à la rencontre de plusieurs restaurateurs afin d’évaluer leurs attentes et leurs avis sur l’application. Dans l’ensemble, l’idée fut bien reçue, mais étant donné que la plupart possédaient déjà une ou plusieurs tablettes, leurs réactions étaient quasiment pareilles pour tous : posséder trop de tablettes était un frein à leur activité et les ralentissaient au lieu de leur faire gagner du temps. Ainsi, je me suis penché vers une technologie peu coûteuse et aisément accessible par le plus grand nombre : le WEB.

Ainsi, j’ai décidé d’utiliser une technologie qui m’était familière en déployant un nouveau projet Symfony sous la version 3.4 à nouveau. Les interfaces déjà réalisées pour Android n’engendraient pas forcément une perte de temps puisqu’elles m’ont permis de travailler l’expérience utilisateur et j’ai pu me forger un premier avis sur les demandes des restaurateurs. ***Voir annexe 5 – Maquette de l’application sous Android***

#### Gestion de l’espace membre

FriendsOfSymfony proposent le bundle Symfony « FOSUserBundle » permettant la gestion de comptes utilisateurs (inscription, connexion, déconnexion, consultation de profil) sécurisée et simple à utiliser. Pour mettre en place ce système, j’ai simplement eu à importer le bundle grâce à composer, puis de configurer Symfony de manière qu’il puisse utiliser le bundle (dans le fichier app\config\config.yml et app\AppKernel.php – ***Voir annexe 6 – configuration de FOSUserBundle***). Une fois la configuration mise en place, j’ai eu à choisir la classe utilisateur dont pourrait se servir le bundle pour la connexion : la classe RestaurantUser (représentant les utilisateurs liés à un restaurant), en implémentant l’interface User de FOSUserBundle (et les méthodes liées : getters et setters pour username, mail, plainPassword et password). Ces modifications apportées, le bundle était configuré correctement (n’ayant pas besoin du formulaire d’inscription, je n’ai pas eu à modifier le RegisterForm de FOSUser). Les formulaires fournis par le bundle sont très basiques, c’est pourquoi je les ai modifiés en créant de nouvelles vues héritant des vues de base du bundle, afin que ces derniers collent avec le design de l’application. ***Voir annexe 7 – modification des vues de FOSUserBundle***).

#### Elaboration d’une charte graphique

Pour garder un design uniforme et représentatif du produit, j’ai utilisé le thème UNITED basé sur le framework CSS bootstrap et disponible sur le site bootswatch.com ; de couleur orange foncé il colle parfaitement avec le rendu attendu par les restaurateurs, et étant basé sur bootstrap, il s’adapte à tous types de plateforme s’il est bien utilisé (mobile / tablette / ordinateur / télévision..).

## Suivi des utilisateurs

### Historisation

Pour pouvoir garder une trace de ce que font les visiteurs sur l'interface, et constater les pages qui sont bloquantes ou du moins sur lesquelles ils se sont arrêtés, j'ai mis en place un système d'historique retraçant les pages que ces derniers ont visités, l'heure à laquelle ils y ont accédés ainsi que l'adresse IP avec laquelle ils s'y sont connectés. J'ai élaboré ce système d'abord dans un but commercial pour faire un compte de nos visites et ainsi des statistiques sur nos clients potentiels, mais aussi pour augmenter l'expérience utilisateur des restaurateurs, en constatant les soucis que ces derniers étaient susceptibles de rencontrer.

*Extrait de la table « history »*

<input type="checkbox"/>	1	<b>id</b> 🔑	int(11)
<input type="checkbox"/>	2	<b>user_id</b> 🔑	int(11)
<input type="checkbox"/>	3	<b>url_visitee</b>	varchar(255) utf8_unicode_ci
<input type="checkbox"/>	4	<b>datetime_visite</b>	datetime
<input type="checkbox"/>	5	<b>ip_visiteur</b>	varchar(255) utf8_unicode_ci

### Restriction d'accès

J'ai pu constater en m'entretenant avec mon tuteur, que les restaurants n'étaient pas forcément voués à être gérés par une seule personne, et que bien souvent les tâches que nous proposons de simplifier sont partitionnées afin d'être plus efficaces. C'est pourquoi je me suis attelé à la création d'un système qui permettrait de gérer différents droits d'accès (car un cuisinier par exemple n'a pas besoin d'avoir accès aux mêmes ressources que le gérant du restaurant). Ainsi, je proposais de gérer les différents modules disponibles de notre plateforme, en les considérant comme des groupes de routes. Il a fallu pour cela enregistrer en base toutes les routes disponibles de l'application, et les associer à un groupe de route (informations, commandes, horaires...). Ces groupes de routes étant ainsi reliés à des utilisateurs, le gérant pouvait choisir les routes auxquelles ses employés utilisant l'application auraient accès. Ce système devait pouvoir nous être utile à l'avenir (en cas de problèmes ou de souhait d'un utilisateur, il suffit d'empêcher l'accès à/aux page(s) désirée(s) via un backoffice ou sur la base de données directement sans toucher au code).

## Guide de première visite

Toujours dans l'optique de simplifier la compréhension de notre système pour les restaurateurs, j'ai réalisé un guide (sous la forme d'un robot) expliquant pas à pas les principales fonctionnalités. Lors de la première visite d'un utilisateur, les différents modules accessibles sont expliqués un à un par le biais d'une fenêtre pop-up (composant bootstrap appelé « modal »), que l'utilisateur devra lire et confirmer afin de passer à l'étape (module) suivante. Une fois toutes les étapes confirmées, les utilisateurs du restaurant pourront accéder sans restriction aux zones qu'ils sont autorisés à consulter.

Les restrictions d'accès étant gérées par des black-listage de routes, j'ai dû créer des restrictions pour chaque nouvel utilisateur, afin de bloquer les pages souhaitées et ainsi adopter le comportement souhaité lors de la première connexion.

Pour ce faire, j'ai mis en place un trigger sur la base de données, s'exécutant à l'ajout d'un nouvel utilisateur, qui allait prendre soin de créer des blacklistages de toutes les routes (sauf la page d'accueil) pour ce dernier.

Une fois la partie visitée débloquée, les routes blacklistées sont déverrouillées (supprimées de la base) et les booléens permettant de savoir si telle ou telle étape est validée, sont mis à la valeur « true ».

*Code SQL du trigger permettant le blacklistage des routes pour un nouvel utilisateur*

```
1 CREATE TRIGGER `blockRoutes` AFTER INSERT ON `restaurant_user`  
2 FOR EACH ROW  
3 INSERT INTO route_black_list (user_id, route_group_id) VALUES  
4 (NEW.id, 1),  
5 (NEW.id, 2),  
6 (NEW.id, 3),  
7 (NEW.id, 4),  
8 (NEW.id, 5),  
9 (NEW.id, 6)|
```

## Module de gestion des informations d'un restaurant

### Connexion d'un utilisateur

Pour que les utilisateurs de la plateforme (restaurateurs) aient accès à la gestion de leur restaurant, carte... il fallait un formulaire de connexion sécurisé. Pour ce faire, FOSUserBundle propose un formulaire de connexion par défaut intégrant des modules sécurisés comme la fonction « se souvenir de moi », très pratique pour garder les identifiants de la personne en mémoire dans ses cookies.

J'ai ainsi modifié le template par défaut de FOSUserBundle (très basique d'un point de vue design), afin de proposer aux utilisateurs un design cohérent avec le reste de l'application.

*Formulaire de connexion d'un employé, modifié de FOSUserBundle.*



The image shows a web form for login. At the top, there is an orange header bar with the word 'Connexion' in white. Below this, the form has a light gray background. It contains two input fields: the first is labeled 'Nom d'utilisateur' and contains the text 'lolo'; the second is labeled 'Mot de passe' and contains a series of dots. Below the password field is a checkbox labeled 'Se souvenir de moi'. To the right of the form is a green button with the text 'S'identifier' in white.

### Gestion des coordonnées

Pour que les apporteurs d'affaires puissent livrer les informations d'un restaurant à leurs clients, les restaurateurs doivent fournir ces informations par le biais d'un formulaire accessible directement sur la page d'accueil vers laquelle ils sont renvoyés par défaut une fois la connexion réussie. Ce formulaire permet à l'utilisateur de modifier les coordonnées du restaurant (Nom, adresse postale, ville, numéro de téléphone, e-mail...). Ces informations permettent aux apporteurs d'affaire de contacter les restaurants, et de les localiser. Le formulaire permet aussi d'associer un logo / une image au restaurant, qui sera envoyée aux applications afin que les clients puissent le reconnaître d'un simple coup d'œil (aussi pour des questions de marketing.). Enfin, il est possible, pour un utilisateur ayant accès à ce module de gérer des informations complémentaires sur le restaurant via ce formulaire telles que le prix moyen du restaurant (économique, moyen, coûteux...) ou encore le type de cuisine via des « tags » (Cuisine Française, Thaïlandaise, Créole...).

### Formulaire de modification des coordonnées d'un restaurant

Modifier les informations de mon restaurant

French Food

Chinese Food

Indian Food

Ajouter un tag

Nom de votre restaurant

Le Port de Lagrange

Adresse du restaurant (ligne 1)

21 rue des Mimosas

Adresse du restaurant (ligne 2)

Adresse du restaurant (ligne 2)

Departement

33 - Gironde

Commune

33310 - LORMONT

Numéro de telephone

055480

Adresse de courrier électronique (e-mail)

Adresse courriel

Numéro IBAN

1854236FRG

Cout estimé

Moyen budget

Description du restaurant

fze

Sauvegarder ces informations

### Gestion des employés

Comme évoqué précédemment, j'ai permis l'accès au cas par cas à tous les modules proposés par notre solution. Ainsi, le gérant peut, par le biais d'un autre formulaire, ajouter des employés comme il le souhaite, en renseignant simplement leur nom d'utilisateur, adresse e-mail et mot de passe (2 fois). En créant ce compte utilisateur, il précise les zones / modules auxquels le futur utilisateur aura accès, ce qui lui permet de gérer ce qui sera accessible aux comptes qu'il vient de créer. Pour avoir accès à ce formulaire, l'utilisateur doit être de rang 2 (c'est le grade correspondant au gérant, il n'y en a qu'un seul par restaurant).

### Formulaire d'ajout d'employés à un restaurant

Ajouter un compte utilisateur

Employee

employe

test

damien

lolo

Adresse e-mail

.....

Sections autorisées

Schedules

Products

DeliveryTowns

Partners

PaymentModes

Orders









Ajouter cet utilisateur

## Module de gestion des horaires

### Services hebdomadaires

Les horaires des restaurants varient rarement d'une semaine à l'autre, et peuvent être découpées en plusieurs plages horaires, aussi appelées services. En partant de ce constat, j'ai mis en place une interface permettant de gérer les différents services en fonction des jours de la semaine. En effet, un restaurant peut, sous la forme de tableau voir les différents services disponibles sur un jour particulier et en ajouter via un formulaire basique contenant des champs de type heure et une liste déroulante contenant les jours de la semaine. Un service ne peut pas faire plus d'1 jour et 2 services ne devaient pas pouvoir se situer sur la même plage horaire. Pour gagner en confort au niveau de l'expérience utilisateur, le restaurant peut modifier les horaires très simplement grâce à un bouton qui fait apparaître un champ de formulaire de type heure grâce à un simple bout de code JQuery qui va envoyer une requête ajax (aussi appelée XMLHttpRequest) de type post vers le Controller Symfony chargé de gérer les horaires. [Voir annexe 14 – Gestion des services](#)

#### *Affichage, modification et suppression des services d'un restaurant*

Les horaires de prise de commande de votre restaurant					
Jour d'ouverture	Service 1		Service 2		Service 3
Lundi	<div>10:30</div> <div>15:00</div>	 	18:00/23:00	 	<a href="#">+ Ajouter un service</a>
Mardi	10:30/15:00	 	<a href="#">+ Ajouter un service</a>		
Mercredi	<a href="#">+ Ajouter un service</a>				
Jeudi	18:00/23:00	 	<a href="#">+ Ajouter un service</a>		
Vendredi	<a href="#">+ Ajouter un service</a>				
Samedi	<a href="#">+ Ajouter un service</a>				
Dimanche	<a href="#">+ Ajouter un service</a>				



#### *Formulaire d'ajout d'un service*

Ajouter un service	
Jour de début du service	<div>Dimanche</div>
Heure de début du service (HH:MM)	<div>--:--</div>
Jour de fin du service	<div>Dimanche</div>
Heure de fin du service (HH:MM)	<div>--:--</div>
<a href="#">+ Ajouter ce service</a>	

## Agenda et horaires particulières

Les restaurants ont effectivement des horaires qui se répètent, ce qui facilite leur gestion, mais ils peuvent parfois faire des exceptions et ouvrir ou fermer pour une période précise. C'est en partant de cette réflexion que j'ai réalisé un module « agenda » permettant de gérer les exceptions dans les horaires. Cet agenda permet à un restaurateur dans un premier temps d'afficher les horaires réelles de son restaurant par rapport à de vraies dates sur un calendrier classé par mois (un restaurant ouvert le lundi verra son restaurant ouvert tous les lundis de l'année sauf exception), et non plus par rapport aux jours de la semaine. Dans un second temps, il peut sélectionner une ou plusieurs dates afin d'y apporter une modification de son statut (Ouvert ou fermé). Ces exceptions sont bien sûr traitées dans l'API, ce qui permet de dire à une application si le restaurant est réellement ouvert ou fermé. Par exemple, si un restaurateur décide de partir en vacances 2 semaines au mois d'août il lui suffit de cocher les dates de ses congés, d'indiquer qu'à ces dates il sera fermé et de sauvegarder ces changements.

### *Gestion de l'agenda d'un restaurant*

 **AVRIL 2018** 

<input type="checkbox"/> Fermé	dimanche 1 avril 2018
<input type="checkbox"/> Ouvert	lundi 2 avril 2018
<input type="checkbox"/> Ouvert	mardi 3 avril 2018
<input type="checkbox"/> Fermé	mercredi 4 avril 2018
<input type="checkbox"/> Ouvert	jeudi 5 avril 2018
<input type="checkbox"/> Fermé	vendredi 6 avril 2018
<input type="checkbox"/> Fermé	samedi 7 avril 2018
<input type="checkbox"/> Fermé	dimanche 8 avril 2018

## Module de gestion de la carte

### Gestion des produits

La carte d'un restaurant un de ses plus gros atout pour attirer les mangeurs, car c'est à cela que ces derniers vont se fier pour commander des produits selon leurs goûts. Cette conclusion tirée, il devenait alors évident que le module de gestion de ses produits serait un de ceux qu'il faudrait le mieux travailler afin de proposer une simplicité d'utilisation et une efficacité défiant toute concurrence. Pour cela, j'ai mis en place un système affichant les produits de la carte triés par catégories de produit. Toutes les catégories étaient visibles par défaut, mais un menu fixe (équipé d'un scroll pour les restaurants possédant une très longue carte) sur la gauche de l'écran permettait d'atteindre rapidement la catégorie souhaitée. Ainsi, les restaurants pouvaient à leur guise modifier ou supprimer des produits très rapidement. Un formulaire basique, créé grâce à l'utilitaire de génération de formulaire de Symfony (avec la commande `doctrine:generate:form <entité>`), permet d'ajouter un produit à la carte très simplement.

Le module permet aussi au restaurateur (lors de la modification d'un produit) de signaler d'éventuelles ruptures de stock, et de désactiver ce produit à la commande, en le signalant aux applications lorsqu'elles réclament les informations de la carte.

En termes de visibilité et d'attractivité, les produits de la carte se devaient de posséder une image, c'est pourquoi il est possible d'en charger une pour chacun de ces produits. Ces images seront bien sûr renvoyées lorsqu'une application souhaitera consulter la carte d'un restaurant.

Entrées (Les entrées du chef)						
Nom	Description	Etat	Propriétés	Prix TTC		
	Salade Niçoise	Laitue, tomates, basilic, oeufs durs, thon, croutons.	Rupture	11.00€		
Ajouter un produit à cette catégorie						
Plats (Les plats de la carte)						
Nom	Description	Etat	Propriétés	Prix TTC		
	Entrecôte (500g) de charolais, servi avec frites ou riz & salade	Rupture	Taille - Sauce - Cuisson -	16.00€		
	Magret de canard avec sa sauce au poivre, accompagné de frites	Disponible	Sauce - Cuisson -	17.50€		
Ajouter un produit à cette catégorie						

Menu dynamique de la carte, suivant le scroll de la page

Entrées	1
Plats	2
Pizzas	4



## Gestion des options/suppléments

Afin de proposer un système viable dans le temps et automatisé le plus possible, chaque produit pouvait posséder des « propriétés », organisés par groupes. Ces groupes de propriétés représentent les options/paramètres possibles pour un produit (Taille, sauce, pâte...), et peuvent être à choix unique ou à choix multiples (par exemple dans un kebab, on ne choisit qu'un seul pain, tandis qu'on peut choisir plusieurs garnitures : salade, tomate, oignons, olives...). Il est possible de préciser un prix différent pour chacune de ces options (une grande pizza coûtera par exemple certainement plus cher qu'une moyenne ou une petite). Ayant remarqué que la plupart des applications offraient la possibilité d'ajouter des suppléments sur certains produits (tels que les burgers, pizzas...), je me suis empressé de créer une interface permettant la gestion de ces suppléments et l'association de ces suppléments à un produit.

Ainsi le formulaire d'ajout/modification d'un produit (il s'agit du même formulaire) se décompose en 3 étapes : Informations du produit, association (+ajout) des propriétés, ainsi que des suppléments.

### Formulaire d'ajout de propriétés de produits

**Les propriétés de produits de votre restaurant**

Type de pâte Taille Sauce Cuisson zflej testeuh test2 zeuifhz

**Ajouter des propriétés**

☐ Propriétés à choix unique

Nom du groupe de propriétés (Taille, Sauce..)

Nom de la propriété (Petite, Grande, Mayonnaise, Ketchup..)

0

€

Nom de la propriété (Petite, Grande, Mayonnaise, Ketchup..)

0

€

+ de propriétés Ajouter ce groupe

### Formulaire d'ajout de suppléments

**Les catégories de suppléments de votre restaurant**

Suppléments Pizza fzeoihoi

**Ajouter des suppléments**

Nom du groupe de suppléments (Suppléments pizza, suppléments salade...)

Nom du supplément (Reblochon, olives, anchois...)

0

€

Nom du supplément (Reblochon, olives, anchois...)

0

€

+ de suppléments Ajouter ce groupe

## Module de gestion des zones de livraison

Certaines sociétés apportent une affaire pour les restaurants possèdent leurs propres livreurs et/ou sous-traitent ce service à des sociétés externes. Cependant, certaines référencent aussi des restaurants qui, eux même, possèdent leur propre service de livraison (c'est le cas en général pour des restaurants tels que les pizzerias). Il fallait donc prendre en compte ces cas dans notre service afin d'assurer une compatibilité avec toutes les applications et tous les restaurants. C'est pourquoi j'ai mis en place un module où le restaurant pouvait choisir les types de commandes qu'il était prêt à recevoir (Livraison par lui-même, livraison par une société tierce, à emporter...). Un restaurant acceptant de prendre en charge des commandes qu'il livrera lui-même devait donc pouvoir saisir ses zones de livraison. Même si la plupart des services de livraisons permettent la livraison sur toute une zone d'un code postal, j'ai préféré voir le plus précis possible en divisant les codes postaux en communes (possédant elles-mêmes un code INSEE). J'ai donc rempli notre base de données avec la liste des départements (français pour le moment) et une liste des communes (françaises aussi) possédant chacune un département, un code postal (pouvant être commun), ainsi qu'un code INSEE (unique).

J'ai donc créé un formulaire permettant de renseigner les zones de livraisons d'un restaurant (accessible uniquement aux restaurants acceptant de délivrer des commandes eux-mêmes). L'utilisateur pouvait ainsi saisir un code postal depuis lequel il souhaitait pouvoir recevoir des commandes, déclenchant, si le code était valide, l'apparition d'une liste déroulante des communes françaises. Le restaurateur avait aussi la possibilité, pour une zone de livraison de préciser le prix minimum de la commande pour être livrée sur cette commune, ainsi que les frais associés à cette livraison. ***Voir annexe 18 – Formulaire d'ajout des zones de livraison d'un restaurant***

### Formulaire d'ajout et affichage des zones de livraison d'un restaurant

**Ajouter des zones de livraison**

Aucune ville ne possède ce code postal

▼

**Les zones que je peux livrer**

MERIGNAC (33700) Min. : 15 € - Frais : 5 € 

BEGLES (33130) Min. : 20 € - Frais : 2.5 € 

LORMONT (33310) Min. : 25 € - Frais : 3.5 € 

CENON (33150) Min. : 25 € - Frais : 3.5 € 

## Module de gestion des moyens de paiement

Pour chacun des types de commandes possibles, les restaurateurs peuvent prendre en charge différents moyens de paiement (Carte bleue, espèces, chèque déjeuner, chèque vacances...). Par exemple un restaurateur prenant en charge la livraison par lui-même n'aura peut-être pas une machine à carte bleue mobile que le livreur pourra transporter. Ainsi, il fallait que le restaurateur puisse décider de quels moyens de paiements (que j'ai saisi au préalable dans la base de données) il pourrait prendre en charge pour tel ou tel type de commande. Pour ce faire, j'ai conçu un formulaire très simple qui, pour chaque type de commande permet de cocher des « checkbox » correspondantes à chacun des moyens de paiement. Pour simplifier ce formulaire et une fois de plus faire gagner du temps à nos utilisateurs, le formulaire est équipé sur chacun des types d'une checkbox particulière permettant de cocher ou de décocher tous les moyens de paiement.

### *Choix des moyens de paiement du restaurant*

Sur Place		<input checked="" type="checkbox"/> Tout cocher
<input checked="" type="checkbox"/> Carte VISA - MASTERCARD	<input type="checkbox"/> Carte Electron	
<input checked="" type="checkbox"/> Chèque	<input checked="" type="checkbox"/> Virement bancaire	
<input checked="" type="checkbox"/> Espèces		

Livraison par le restaurant		<input checked="" type="checkbox"/> Tout cocher
<input checked="" type="checkbox"/> Carte VISA - MASTERCARD	<input checked="" type="checkbox"/> Carte Electron	
<input checked="" type="checkbox"/> Chèque	<input type="checkbox"/> Virement bancaire	
<input checked="" type="checkbox"/> Espèces		

Livraison par un tiers		<input checked="" type="checkbox"/> Tout cocher
<input type="checkbox"/> Carte VISA - MASTERCARD	<input type="checkbox"/> Carte Electron	
<input type="checkbox"/> Chèque	<input type="checkbox"/> Virement bancaire	
<input type="checkbox"/> Espèces		

Sauvegarder

## Module de gestion des partenariats

Une autre partie délicate, en raison des poids juridiques et financiers qu'elle engrange, fut la gestion des contrats entre les sociétés tierces et les restaurants. En règle générale, un restaurateur passe un contrat avec la société pour un temps défini, c'est le schéma que j'ai essayé de reproduire dans mon système de gestion des partenariats. En effet, un apporteur d'affaire peut proposer un contrat (cette partie sera développé dans le chapitre sur le backoffice des apporteurs d'affaires) à un restaurant. J'ai ainsi créé sur l'interface des restaurateurs un récapitulatif de ces contrats, avec un petit résumé du contrat, un lien vers le fichier PDF du contrat ainsi qu'un bouton permettant l'acceptation ou le retrait d'un contrat par le restaurateur. Un contrat ne concerne qu'un seul type de commande, et doit être accepté par le restaurateur concerné pour que l'application l'ayant émis puisse envoyer des commandes au restaurant par le biais de l'API.

### *Gestion des partenariats d'un restaurant*

#### Les partenariats disponibles

##### Sur Place

LesRestausQuiLivrent

Nom du contrat	Début le	Prend fin le	Comission		
Contrat a 14	01-04-2018	31-07-2018	14 %	<a href="#">Selectionner</a>	<a href="#">Lire le contrat</a>
Contrat été premium	23-04-2018	09-07-2018	10 %	<a href="#">Selectionner</a>	<a href="#">Lire le contrat</a>

#### Livraison par le restaurant

LesRestausQuiLivrent

Nom du contrat	Début le	Prend fin le	Comission		
Contrat Livraison 3 premiers mois	25-04-2018	31-12-2099	10 %	<a href="#">Rompre le contrat</a>	<a href="#">Lire le contrat</a>

## Module de gestion des commandes

### Suivi des commandes en cours

Il s'agissait là encore d'une partie très importante pour le restaurateur, puisque c'est l'écran qui lui permettait de suivre en tant réel les commandes passées par les diverses applications reliées à notre système. J'ai beaucoup travaillé de manière à lire les commandes le plus simplement et le plus rapidement possible, il m'a fallu pour cela trier quelles informations d'une commande étaient réellement importantes à afficher pour le restaurateur. Cet écran comportait donc évidemment la liste des produits commandés (avec leurs propriétés et suppléments), la référence unique (afin de se retourner vers nous ou l'application le plus rapidement possible en cas de problème et éviter d'éventuels désagréments pour les clients), la zone où cette dernière doit être livrée, ainsi que le montant à encaisser par le restaurateur (pour les cas particuliers où le client souhaiterait payer une partie en espèces et une autre en carte bleue, par exemple), et l'heure à laquelle cette commande doit être prête.

Cette interface devait aussi permettre au restaurateur le suivi de la commande en temps réel, afin d'indiquer au client où elle en est. Les commandes possédaient donc un statut particulier (commandée, vue et acceptée par le restaurateur, en cours de préparation, prête, livrée...). Ce statut est changeable sur l'interface par plusieurs petites manipulations : le restaurateur doit d'abord accepter la commande avant de voir son contenu en intégralité, ensuite il doit checker les produits un à un pour indiquer qu'ils sont prêts. Une fois tous les produits cochés, la commande est signalée comme prête et le bouton « envoyée » est déverrouillé pour le restaurateur, qui n'a plus qu'à cliquer dessus pour retirer la commande de l'écran et signaler aux applications (et donc aux clients) que la commande vient d'être envoyée. Une amélioration à laquelle j'ai pensé est d'offrir au restaurateur la possibilité de signaler qu'un produit est en rupture de stock, après que la commande soit passée par un client, c'est une fonctionnalité que je réserve pour la version 2 de l'application. Le détail de chaque commande peut être affiché sur une page sous la forme d'un ticket de caisse imprimable, dont j'ai pris soin d'adapter le design à l'aide d'une feuille de style CSS dédiée uniquement à l'impression sur une imprimante spécialisée pour les tickets de caisse (la page s'adapte donc à un format miniature).

### Suivi des commandes en cours

Commandes à prendre

Historique des commandes

Temps de préparation d'une commande (HH:MM)

00:55

Temps de livraison d'une commande (HH:MM)

01:05

Valider ces infos.

Commande : AZW1IRS9AJ

Nom du produit	Propriétés	Suppléments	Précisions	Prix TTC	
Margherita	- Pâte Fromagère - Grande			15.00 €	<input type="checkbox"/> Prêt
Margherita	- Pâte Fromagère - Grande			15.00 €	<input type="checkbox"/> Prêt
				31.46 €	32.50 €

33130 BEGLES

Zone de livraison

12h50

Heure de livraison

24.00 €

A encaisser

✓ Commande envoyée

Commande : 1MZ05QBWHY

J'ai vu et accepte la commande

Nom du produit	Propriétés	Suppléments	Précisions	Prix TTC	
Margherita	- Pâte Fromagère - Grande			15.00 €	<input type="checkbox"/> Prêt
Margherita	- Pâte Fromagère - Grande			15.00 €	<input type="checkbox"/> Prêt
				31.46 €	32.50 €

33130 BEGLES

Zone de livraison

12h50

Heure de livraison

24.00 €

A encaisser

✓ Commande envoyée

## Historisation

L'application devait aussi pouvoir proposer aux restaurateurs un historique des commandes qui lui ont été passées, afin que ces derniers puissent d'une part retracer les produits qu'ils ont écoulés, et ainsi les aider à gérer leurs stocks et leur comptabilité, mais aussi les aider en cas de litige à justifier leurs revenus sur les commandes et ainsi veiller à la bonne mise en pratique du contrat passé avec les sociétés externes.

J'ai représenté cet écran sous la forme d'un tableau, où apparaissent les informations sur la société ayant passé la commande, l'état courant de la commande, ainsi que les diverses informations concernant la partie comptabilité (total HT, total TTC, montant encaissé par le restaurant, montant de la commission pour la société...). A la fin de ce tableau, j'ai fait afficher le total TTC des revenus du restaurant. ***Voir annexe 23 – Historique des commandes d'un restaurant***

Cet écran concerne encore à l'heure actuelle toutes les commandes, ce qui est suffisant pour l'instant au vu du peu de commandes passées sur l'application, mais j'ai l'intention d'y ajouter des filtres (par moi, trimestre, société de livraison...), afin que le restaurateur puisse trier les commandes qu'il a reçues et ainsi faire ses comptes de manière mensuelle, trimestrielle.... Cette amélioration augmentera grandement la simplicité d'utilisation et pourrait faire office de solution comptable pour le restaurant, à supposer que toutes ses commandes passent par notre réseau.

### *Historique des commandes du restaurant*

Récapitulatif des commandes							
Reference	Nom de la société	Perçu par moi	Percu par la société	Comission	Total HT	Total TTC	Statut
1MZO5QBWHY	LesRestausQuiLivrent	24.00 €	08.50 €	10.00 €	31.46 €	32.50 €	22:52 - Ordered
AZW11RS9AJ	LesRestausQuiLivrent	24.00 €	08.50 €	10.00 €	31.46 €	32.50 €	20:14 - Seen
HI30DENSTI	LesRestausQuiLivrent	24.00 €	-4.00 €	10.00 €	19.48 €	20.00 €	11:48 - Taken
EBKH6FKXIMZ	LesRestausQuiLivrent	24.00 €	26.00 €	10.00 €	48.44 €	50.00 €	11:55 - Taken
GPYJK9UFOX	LesRestausQuiLivrent	24.00 €	11.00 €	10.00 €	33.96 €	35.00 €	12:01 - Taken
D63SHAZ8HU	LesRestausQuiLivrent	24.00 €	11.00 €	10.00 €	33.96 €	35.00 €	12:02 - Taken
L2N0T6BG1E	LesRestausQuiLivrent	10.00 €	14.50 €	05.00 €	23.74 €	24.50 €	15:03 - Taken
GT2NWLURGT	LesRestausQuiLivrent	19.00 €	25.00 €	07.00 €	42.48 €	44.00 €	11:04 - Taken
A7W1MZK1QR	LesRestausQuiLivrent	24.00 €	39.50 €	10.00 €	61.22 €	63.50 €	11:35 - Taken
JO5Y7C9YFW	LesRestausQuiLivrent	24.00 €	00.50 €	10.00 €	23.74 €	24.50 €	11:23 - Taken
VOL2B8HINS	LesRestausQuiLivrent	10.00 €	12.50 €	05.00 €	21.59 €	22.50 €	13:03 - Taken
						36.6 €	

## V – Conception d'un backoffice pour les applications

L'objectif de ce projet était de proposer aux utilisateurs (« marketplaces ») une interface de gestion de leurs relations avec les restaurants, en vue de gérer leur compte utilisateur du Webservice dédié au transit des informations par leurs applications.

L'application se découpait en plusieurs modules permettant de :

- Voir la liste des restaurants partenaires, filtrés par départements
- Créer des contrats par types de commandes (Livraison par l'application, livraison par le restaurant, sur place..)
- Proposer des contrats à un ou plusieurs restaurants
- Gérer les informations de son compte
- Gérer son solde

Pour répondre à une contrainte de temps très légère, j'ai créé cette plateforme en utilisant les technologies qui me sont les plus familières :

- Le langage PHP natif (sans framework)
- Le framework CSS bootstrap équipé du même thème (issu du site [bootswatch.com](https://bootswatch.com)) que la partie restaurateurs
- Les langages de mise en forme HTML / CSS et Javascript ainsi que la bibliothèque jQuery pour faciliter la dynamisation de l'interface.

## Module de gestion des contrats

Pour que les marketplace puissent effectuer toutes les actions nécessaires à leur bon fonctionnement via le Webservice, il fallait que le restaurant et la marketplace aient passés un contrat ensemble, et que ce dernier l'ait accepté. Chaque type de commande possible de notre service (Livraison, emporter...) possédaient des contrats différents (par exemple un restaurant souhaitant travailler avec une marketplace pour recevoir des commandes à emporter ne souhaitera pas forcément que cette marketplace s'occupe de la livraison vers le client...)

J'ai donc dû penser une interface permettant aux utilisateurs de créer de nouveaux contrats, ainsi que de voir les contrats existants (ainsi que des statistiques sur ces derniers), le tout par types de commandes.

Un contrat possède une durée de vie (début de validité, fin de validité), un statut, un type de commandes concernés ainsi que le fichier pdf du contrat écrit.

### Formulaire d'ajout d'un contrat

#### Ajouter un contrat

**Fichier pdf du contrat**

Aucun fichier choisi

### Affichage des contrats

#### Mes contrats par type

##### Sur Place (2)

Contrat a 14  
Comission : 14 %  
1 restaurant(s) concerné(s)

Contrat été premium  
Comission : 10 %  
1 restaurant(s) concerné(s)

##### Livraison par le restaurant (1)

Contrat Livraison 3 premiers mois  
Comission : 10 %  
2 restaurant(s) concerné(s)

##### Livraison par un tiers (0)

Vous n'avez pas encore de contrat pour ce type

##### Accès aux données (0)

Vous n'avez pas encore de contrat pour ce type



## Module de gestion des restaurants partenaires

Ce module devait permettre aux utilisateurs d'avoir une vue d'ensemble sur leurs relations avec nos restaurants partenaires. Pour se faire, j'ai réfléchi à un affichage le plus ergonomique possible : une liste de tous les restaurants partenaires, synthétisant leurs relations. Les utilisateurs peuvent trier ces restaurants à l'aide d'un filtre par département, pour faciliter la lecture au vu du nombre de restaurants partenaires ambitionnés (+ de 10 000).

Une checkbox à côté de chaque restaurant permettait aux utilisateurs de les sélectionner afin de leur proposer un même contrat.

### *Affichage de la liste des restaurants*

The screenshot shows a web interface titled "Les restaurants de CMR". It features a "Filtres" section with a dropdown menu set to "Tous les dpts.". Below this, there is a table of restaurants. The first row is "Le Port de Lagrange" with "3 contrat(s) proposés(s)" and a "Voir le restaurant" button. The second row is "RESTAU SANS CONTRAT" with "1 contrat(s) proposés(s)" and a "Voir le restaurant" button. At the bottom, there is a green button labeled "Proposer un contrat aux restaurant sélectionnés".

Les restaurants de CMR		
Filtres		
Tous les dpts.		
<input type="checkbox"/> Tout sélectionner		
<input type="checkbox"/> Le Port de Lagrange	3 contrat(s) proposés(s)	<a href="#">Voir le restaurant</a>
<input type="checkbox"/> RESTAU SANS CONTRAT	1 contrat(s) proposés(s)	<a href="#">Voir le restaurant</a>
<a href="#">Proposer un contrat aux restaurant sélectionnés</a>		

### *Proposition de contrat aux restaurants sélectionnés*

The screenshot shows a web interface titled "Proposition de contrat". It displays the text "Le contrat sélectionné sera proposé aux restaurants suivants : - Le Port de Lagrange - RESTAU SANS CONTRAT". Below this, there is a section "Contrat à proposer" with a dropdown menu set to "Contrat a 14". There are two date input fields labeled "Valide du" and "au", both with the placeholder "jj/mm/aaaa". At the bottom right, there is a green button labeled "Selectionner ce contrat".

Proposition de contrat	
Le contrat sélectionné sera proposé aux restaurants suivants : - Le Port de Lagrange - RESTAU SANS CONTRAT	
Contrat à proposer	
Contrat a 14	
Valide du	au
jj/mm/aaaa	jj/mm/aaaa
<a href="#">Selectionner ce contrat</a>	

## Module de gestion du compte

### Gestion des informations du compte API

Les utilisateurs possédant un compte sur le Webservice devaient pouvoir consulter les informations sur leur société, ainsi que des statistiques rapides pour pouvoir avoir un bilan global sur leur activité d'un simple coup d'œil. Ils devaient aussi pouvoir générer une nouvelle clé d'API, en cas de vol ou de compromission de la confidentialité de cette dernière. J'ai donc créé comme écran d'accueil une partie résumant toutes ces informations.

*Ecran d'accueil, résumé des informations d'un utilisateur API*

Mes informations	
<b>LesRestausQuiLivrent</b>	<b>loicCMR</b>
Mon Solde	loic.sicaire@test.com
400.3 € (soit ~ 1201 commandes à 30€)	21 Rue Hector Berlioz, 33130 Bègles
Clé d'API	3 contrat(s) disponible(s)
ZCltbH8v/SLvEC7/y0UPIOnenuw=	1 restaurant(s) partenaire(s) / 4 contrat(s) proposés
<a href="#">Changer ma clé d'API</a>	

### Gestion du solde d'un utilisateur

Les utilisateurs devaient posséder un solde, leur permettant de passer des commandes vers un restaurant, en versant une comission d'un montant de 1% de la commande. Pour réaliser un tel système, j'ai d'abord examiné les différentes API de paiement en ligne (les systèmes proposés actuellement par les banques étant trop onéreux en termes de gestion et de finances).

J'ai donc fais un choix entre les APIs Stripe et MangoPay. J'ai choisi l'API Stripe, plus simple d'utilisation, plus en adéquation avec nos attentes et moins couteuse, malgré que j'ai déjà eu l'occasion d'utiliser MangoPay dans des projets antérieurs. Stripe est une API permettant de gérer des paiement dans plusieurs devises et plusieurs pays, sans se soucier trop des législations en vigueur dans ces pays (ils s'en occupent eux-mêmes).

J'ai donc installé un SDK (Software Development Kit) PHP qu'ils proposaient pour simplifier l'intégration de l'API à notre système. Les extraits de codes proposés étant très simple à intégrer, je n'ai pas mis très longtemps avant de pouvoir faire mes premiers paiements sur le serveur de test et à intégrer la solution à l'espace marketplace.

L'interface que j'ai créée proposait donc aux utilisateurs, conformément à l'utilisation préconisée, de rajouter du crédit à leur solde en effectuant une transaction par carte bancaire sur notre compte Stripe. Le formulaire de paiement que j'ai créé propose soit de payer directement avec une nouvelle carte bancaire en saisissant ses informations (16 chiffres / date de validité / cryptogramme visuel), soit avec une carte qu'ils ont déjà saisie sur nos serveurs.

*Formulaire de paiement permettant l'ajout de crédit à son solde*

Créditer mon compte

Créditer avec une carte existante

Montant à créditer

30.0

Carte sélectionnée

XXXX XXXX XXXX 4242 - 4 / 2024

Valider et payer

Créditer avec une nouvelle carte

## Module de visualisation d'un restaurant

### Historique et passage de contrats

Sur la liste des restaurants, un bouton sur chacun d'entre eux mène vers l'écran où figurent les détails des relations avec ce restaurant. En effet je devais fournir aux utilisateurs un historique complet des partenariats qu'il a passé avec ce restaurant. Ainsi figurent sur cette page tous les contrats, avec leur statut (contrat accepté, refusé ou en attente de réponse). Le statut est accompagné d'un code couleur pour le distinguer au premier coup d'œil.

Pour qu'un contrat soit valide et considéré comme accepté, il faut qu'il soit encore en cours de validité, et que le restaurateur l'ait accepté et n'y ait pas mis un terme. (En clair il faut que le contrat soit encore dans une période de validité active et que les deux acteurs soient toujours en accord).

Les contrats actifs peuvent être désactivés par l'utilisateur ou par le restaurateur, et les contrats désactivés par l'utilisateur peuvent être reproposés par le biais de cette interface.

J'ai aussi dans ce module offert la possibilité de proposer un nouveau contrat, parmi les contrats créés de sorte à pouvoir proposer des contrats à un restaurant en particulier pour un type de commandes particulières.

### Affichage de l'historique des contrats avec un restaurant particulier

#### Les contrats de ce restaurant

##### Sur Place

Contrat a 14

Valide du 01-04-2018 au 31-07-2018

Actif du : 25-04-2018 au : 25-04-2018

Actif du : 25-04-2018 au : 25-04-2018

Contrat été premium

Valide du 23-04-2018 au 09-07-2018

##### Les contrats disponibles pour ce type

Contrat a 14

Valide dujj/mm/aaaaaujj/mm/aaaa

Selectionner ce contrat

##### Livraison par le restaurant

Contrat Livraison 3 premiers mois

Valide du 25-04-2018 au 31-12-2099

Actif du : 25-04-2018 au : 31-12-2099

Contrat Livraison 3 premiers mois

Valide du 18-05-2018 au 25-05-2018

##### Les contrats disponibles pour ce type

Contrat Livraison 3 premiers mois

Valide dujj/mm/aaaaaujj/mm/aaaa

Selectionner ce contrat

##### Livraison par un tiers

Aucun contrat actuellement proposé pour ce type

Pas de contrat à proposer pour ce restaurant

Ajouter des contrats

##### Accès aux données

Aucun contrat actuellement proposé pour ce type

Pas de contrat à proposer pour ce restaurant

Ajouter des contrats

## Historique des commandes

J'ai par la suite développé une seconde partie dans laquelle l'utilisateur peut voir l'historique des commandes qu'il a passées à ce restaurant, pour pouvoir faire par la suite un bilan de ce que ce qu'il a apporté au restaurant et gérer d'éventuels conflits plus simplement.

Les commandes peuvent être filtrées par statut de manière à libérer un peu l'affichage si l'écran est trop chargé.

### *Affichage des commandes d'un utilisateur à un restaurant particulier*

Les commandes du restaurant

Tri des commandes

Trier par statut :  

Ordered

Commande 1MZO5QBWHY

Comission : 10 €      Date : 2018-05-28 12:50:00      Ordered

Commande AZW1IRS9AJ

Comission : 10 €      Date : 2018-05-28 12:50:00      Ordered

## VI – Améliorations possibles

La version 1 de chacune des applications de ConnectMyResto a été déployée sur le serveur de production en Mai 2018, et semble contenter les utilisateurs depuis. Cependant, le mot d'ordre de CMR étant l'amélioration continue, certains modules / parties de programme nécessiteraient de petits ajustements, et les applications en général sont vouées à être maintenues dans le temps, et à faire l'objet d'ajout de fonctionnalités.

### WebService

Concernant l'API que j'ai développé, je pense qu'il s'agit là du point central du projet, elle possède des fonctionnalités déjà bien étoffées. Ce qui n'empêche que certains traitements (notamment ceux de la liste des restaurants ainsi que le compte des restaurants par département/pays) pourraient être plus rapide avec des optimisations dans les algorithmes.

Aussi, il m'est venu l'idée de rajouter certains filtres dans les requêtes, que je n'ai pas eu le temps de développer pour la version 1 (notamment la possibilité de préciser soit un code INSEE, soit le nom et code postal d'une ville dans la requête de recherche de restaurants en livraison).

Une requête potentiellement très utile risque fort de voir le jour dans les prochaines versions : une requête permettant de récupérer la liste des communes d'un code postal, avec leur code INSEE, de manière à ce que les utilisateurs de l'API n'aient plus à connaître les codes INSEE des villes qu'ils souhaitent faire livrer, comme c'est le cas aujourd'hui.

L'idée même de l'architecture MVC laisse à penser que cette application se maintiendra assez durablement dans le temps, c'est d'ailleurs une des raisons pour lesquelles j'ai choisi d'utiliser cette technologie. De surcroît la version 3.4 de Symfony est toujours maintenue à l'heure où j'écris ces lignes.

### Interface de gestion restaurateurs

L'interface pour les restaurateurs étant développée en Symfony, une des grosses erreurs que j'ai commises d'après moi a été de reprendre un projet Symfony en calquant mes entités sur celles déjà existantes dans le WebService, au lieu d'utiliser ce dernier (voire, pour plus de sécurité, une nouvelle API, développée pour les opérations internes) comme modèle. Cela a eu pour incidence un double travail pour le maintien des 2 apps ensembles, puisque modifier la structure de l'une ne peut se résulter que par une modification de l'autre (pas très souple pour un projet d'une telle taille).

La version 2 de cette application sera très probablement calquée sur la même identité graphique, mais le back-end lui sera le WebService précédemment cité. Ainsi, un seul et même back servira pour cette interface, et pour toutes les applications futures, ce qui assurera un maintien à jour bien plus simple à gérer.

## Back-office marketplaces

J'ai conçu le back-office pour les marketplaces à la hâte, en raison de la demande imminente qui m'a été faite par les utilisateurs du Webservice, c'est pourquoi j'ai employé la technologie que je maîtrise probablement le mieux pour l'avoir pratiqué pendant nombre d'années : le développement web basique de manière procédurale et sans utiliser de Framework PHP.

Ce choix fut à double tranchant, puisqu'il avait l'avantage d'être rapide à mettre en place, mais peu maintenable dans le temps par rapport à la structure globale de notre application (une simple modification ou ajout d'un champ en base de données peut entraîner nombre de changements dans le code). C'est pourquoi, comme pour l'interface des restaurateurs, je prévois dans une version suivante de refaire le backend de l'application, avec uniquement des appels vers l'API interne que je compte créer d'ici là.

De plus, certaines demandes m'ont été transmises de la part des utilisateurs, de rajouter des modules pour l'ajout de restaurants par un utilisateur de l'API (si l'utilisateur démarche un restaurant, il voudrait probablement l'ajouter à notre base sans passer par un contact avec nous), ainsi que le passage de commandes à un restaurant, via l'interface graphique, si une commande spontanée à lieu ou en cas de bug de l'application du marketplace.

Enfin, j'ai prévu d'ajouter des filtres pour trier les restaurants (par contrats, ville, statuts de contrat, types de cuisine...) et pour trier les commandes afin d'étoffer encore la fluidité de ces écrans de listing.

## Infrastructure globale

Un gros problème pour le développement de ConnectMyResto est son infrastructure globale, très peu adaptée aux ambitions prêtées au départ. En effet, l'hébergement mutualisé et les bases de données limitées à 1Go de stockage ne seront certainement pas suffisantes pour exécuter les futures requêtes des applications et stocker les milliers de restaurants comme nous souhaitons le faire.

Afin d'offrir une plus grande disponibilité, une bonne répartition des charges et un meilleur service en général à nos utilisateurs, nous prévoyons de posséder notre propre infrastructure sécurisée pour héberger ces applications couteuses en ressources.

Au vu du nombre très important de données à stocker, il sera peut-être judicieux de s'orienter vers des technologies plus adaptées au « big data », type no-SQL ou autres.

Enfin, d'autres applications devraient voir le jour, toujours dans la même optique (notamment la version mobile des applications existantes. Bien que déjà web-responsive, une application mobile sera plus à même de proposer un service adéquat aux utilisateurs souhaitant utiliser leur tablette / téléphone pour gérer leurs comptes).

## VII – Conclusion

Avec le recul, je peux dire que ce projet m’a fortement plu, et que cette année m’a beaucoup appris, non seulement en termes de compétences techniques (je ne savais pas comment fonctionnait une API et avait reçu une vingtaine d’heures de formation sur Symfony avant de rentrer chez Mate-maker), mais aussi en termes de compétences professionnelles et entrepreneuriale. En effet, j’ai eu l’occasion d’être confronté à de vraies responsabilités quant aux choix stratégiques de l’entreprise, en tant qu’alternant dans un premier temps, mais aussi désormais en tant qu’associé.

L’aventure ConnectMyResto m’a beaucoup apporté sur des tas d’aspects : j’ai pu me mettre dans la peau d’un chef de projet durant un an et je tiens à remercier pour cela mon tuteur en entreprise, Damien Huctin, fondateur de la société Mate-maker, qui m’a énormément appris sur le monde de l’entrepreneuriat et ses valeurs.

Etant présent depuis la naissance de l’idée, ce projet me tient beaucoup à cœur. Je souhaiterais le voir évoluer le plus possible, et y apporter ma contribution jusqu’à en être totalement satisfait.

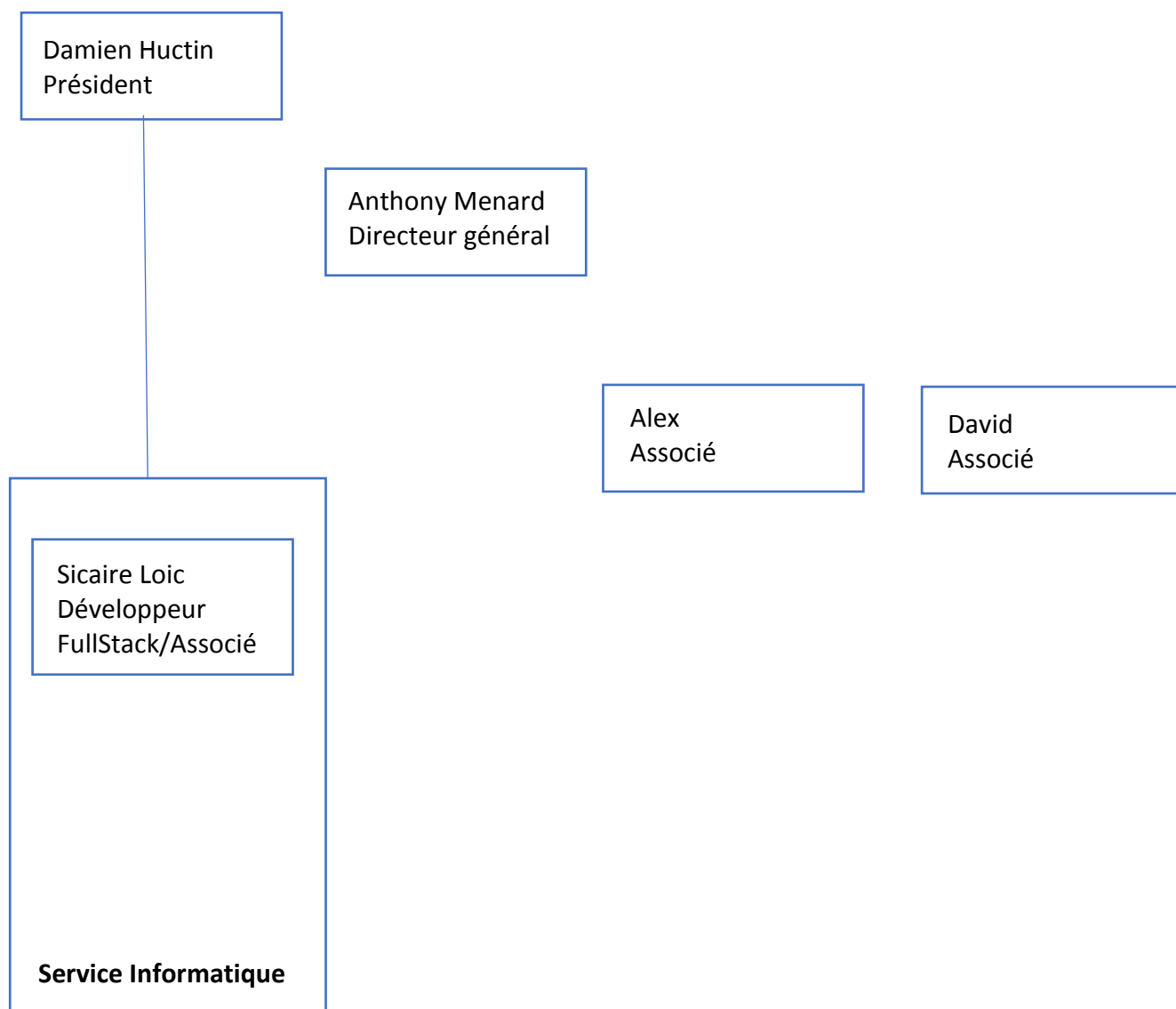
Sur un plan personnel, cette année m’a permis de confirmer mes ambitions de créer ma propre start-up, dans le but d’apporter des solutions viables à des problèmes existants.



## Annexe 1 – Liste des compétences couvertes par le projet

Activités type	Compétences	
Développer des composants d'interface	Maquetter une application	X
	Développer une interface utilisateur	X
	Développer des composants d'accès aux données	X
	Développer des pages web en lien avec une base de données	X
Développer la persistance des données	Concevoir une base de données	X
	Mettre en place une base de données	X
	Développer des composants dans le langage d'une base de données	X
	Utiliser l'anglais dans son activité professionnelle en informatique	X
Développer une application n-tiers	Concevoir une application	X
	Collaborer à la gestion d'un projet informatique	
	Développer des composants métier	X
	Construire une application organisée en couches	X
	Développer une application de mobilité numérique	X
	Préparer et exécuter les plans de tests d'une application	
	Préparer et exécuter le déploiement d'une application	

## Annexe 2 – Organigramme de la société



## Annexe 3 – Classe AuthTokenAuthenticator

```
\AppBundle\Security AuthTokenAuthenticator

49         $providerKey
50     );
51 }
52
53 public function authenticateToken(TokenInterface $token, UserProviderInterface $userProvider, $providerKey)
54 {
55     if (!$userProvider instanceof AuthTokenUserProvider) {
56         throw new \InvalidArgumentException(
57             sprintf(
58                 format: 'The user provider must be an instance of AuthTokenUserProvider (%s was given).',
59                 get_class($userProvider)
60             )
61         );
62     }
63
64     $authTokenHeader = $token->getCredentials();
65     $authToken = $userProvider->getAuthToken($authTokenHeader);
66
67     if (!$authToken || !$this->isTokenValid($authToken)) {
68         throw new BadCredentialsException('Clé invalide.');
```

69 }

70

71 \$user = \$authToken->getApiUser();

72 \$pre = new PreAuthenticatedToken(

73 \$user,

74 \$authTokenHeader,

75 \$providerKey,

76 \$user->getRoles()

77 );

78

79 // Nos utilisateurs n'ont pas de rôle particulier, on doit donc forcer l'authentification du token

80 \$pre->setAuthenticated(true);

81

82 return \$pre;

83 }

84

85 public function supportsToken(TokenInterface \$token, \$providerKey)

## Annexe 4 – Création d'une commande

```
/** @Rest\View(serializerGroups={"orders"}) ... */
public function createOrderAction(Request $req){
    if ($req->get( key: "orderType") == null)
        throw new BadRequestHttpException( message: "You need to specify an orderType to send command");

    $em = $this->get('doctrine.orm.entity_manager');

    $type = $req->get( key: "orderType");
    $type = $this->getDoctrine()->getRepository("AppBundle:OrderType")->find($type);
    if (!$type || empty($type))
        throw new BadRequestHttpException( message: "This orderType does not exist");

    $buisness = $this->getBusiness($req);
    $restau = $this->getRestaurant($req->get( key: "id_r"));

    $order = new Order();
    $order->setBusiness($buisness);
    $order->setRestaurant($restau);
    $order->setType($type);
    $order->setReference($this->random( car: 10));

    $this->verifyWorkWith($buisness, $type, $restau);

    $order = $this->setOrderParams($req, $order);
    $em->persist($order);

    $order = $this->setMissingParams($order);
    $em->flush();

    return $order;
}
```

## Annexe 5 – Ajout / suppression d'un produit

### Ajout d'un produit

```
/** @Rest\View(serializerGroups={"orders"}) ... */
public function addProductAction(Request $request){
    $em = $this->get('doctrine.orm.entity_manager');
    $idOrder = $request->get('key: "id_order"');
    $order = $this->getOrder($idOrder);

    // Si la commande est confirmée, erreur 400
    if ($order->status()->getId() > 2)
        throw new BadRequestHttpException( message: "This order is confirmed. You can't add a product, you may create a new order ?!");

    $this->orderVerifications($request, $order);

    // Ajout du produit à la commande
    $order = $this->addOrderProduct($request, $order->getType(), $order);

    // calcul des promotions éventuelles et mise à jour des paramètres manquants
    $order = $this->calculReductions($order);
    $order = $this->setMissingParams($order);

    $em->merge($order);
    $em->flush();

    return $order;
}
```

### Suppression d'un produit

```
/** @Rest\View(serializerGroups={"orders"}) ... */
public function removeProductAction(Request $request){
    $idOrderProd = $request->get('key: "id_p"');
    $idOrder = $request->get('key: "id"');
    $order = $this->getOrder($idOrder);

    // Si la commande est déjà confirmée, erreur 400
    if ($order->status()->getId() > 2)
        throw new BadRequestHttpException( message: "This order is confirmed. You can't add a product, you may want to create a new order ?!");

    $this->orderVerifications($request, $order);

    // Suppression du produit
    $orderProd = $this->getDoctrine()->getRepository("AppBundle:OrderProduct")->find($idOrderProd);
    $order = $this->removeOrderProduct($orderProd, $order);

    $order = $this->calculReductions($order);

    $order = $this->setMissingParams($order);

    return $order;
}
```

## Annexe 6 – Confirmation d’une commande

```
/** @Rest\View(serializerGroups={"orders"}) ...*/
public function confirmOrderAction(Request $request){
    $idOrder = $request->get( key: "id");
    $order = $this->getOrder($idOrder);
    $this->orderVerifications($request, $order);

    // Si la commande est confirmable
    if ($order->getActiveStatus()->getStatus()->getId() == 2) {
        $em = $this->get('doctrine.orm.entity_manager');
        // Confirme la commande
        $this->setStatus($order, idStatus: 3);
        // Notification du restaurant pour une nouvelle commande
        $notif = new Notification();
        $notif->setRestaurant($order->getRestaurant());
        $notif->setText("Une commande vient d'être passée de la part de " . $order->getBusiness()->getName() . ". Venez la consulter !");
        $notif->setIsSeen(false);
        $notif->setIsShown(false);
        $notif->setOrder($order);
        $notif->setSentAt(new \DateTime());

        // débite le compte de l'application
        $this->debitBusiness($order);

        $em->persist($notif);
        $em->merge($order);
        $em->flush();
        return $order;
    }else
        throw new BadRequestHttpException( message: "This order is not Complete yet.");
}
```

## Annexe 7 – Postman par Google / Elaboration d'un jeu de requêtes

The screenshot displays the Postman interface for running a collection. The main window is titled 'COLLECTION RUNNER' and shows the 'API-CMR [Test]' collection. The 'Previous Runs' section lists several successful runs. The 'CURRENT RUN' section shows the details of the current run, including the environment, iteration, and delay. The 'RESULTS' section displays the status of each request in the collection.

**API-CMR [Test]**  
Last Modified: 27 Apr, 2018  
Owner: You

**Previous Runs**

- Api-cmr [test] - No environment, just now - All Passed
- Api-cmr [local] - No environment, a min ago - All Passed
- Api-cmr [local] - No environment, 2 hrs ago - All Passed
- Api-cmr [test] - No environment, 27 Apr, 2018 - All Passed
- Api-cmr [test] - No environment, 27 Apr, 2018 - All Passed
- Api-cmr [test] - No environment, a month ago - All Passed
- Api-cmr [test] - No environment, a month ago - All Passed
- Api-cmr [local] - No environment, 26 Feb, 2018 - All Passed
- Api-cmr [local] - No environment, 26 Feb, 2018 - All Passed
- Api-cmr [local] - No environment, 14 Feb, 2018 - All Passed

**CURRENT RUN**

Search for a collection or folder

< API-CMR [Test]

GET getRestaurants  
GET getRestaurantsByCountryCode  
GET getRestaurantByGPS  
POST createOrder  
POST addOrderProduct  
DELETE removeOrderProduct

Environment: No environment  
Iteration: 1  
Delay: 0  
Data File: Sélectionner fichiers | Aucun fichier choisi  
Persist Variables: ☒

**RESULTS**

0 passed 0 failed 37137 ms

getRestaurantByGPS  
http://api.test.connectmyresto.com/restaurants/take-...  
200 OK  
10356 ms

No tests

createOrder  
http://api.test.connectmyresto.com/restaurants/1/cre...  
200 OK  
275 ms

No tests

addOrderProduct  
http://api.test.connectmyresto.com/orders/154/add-...  
200 OK  
1630 ms

No tests

removeOrderProduct  
http://api.test.connectmyresto.com/orders/154/remo-...  
200 OK  
317 ms

No tests

updateOrder  
http://api.test.connectmyresto.com/orders/154/update...  
200 OK  
595 ms

No tests

confirmOrder  
http://api.test.connectmyresto.com/orders/154/confi-...  
200 OK  
3111 ms

No tests

## Annexe 8 – Documentation de l'API

### Navigation

**Général**

Accueil

Obtenir une Clé d'API

S'identifier

Conditions d'utilisation

**Documentation**

Débuter avec CMR

**Restaurants ▾**

Structure de l'objet

Récupérer la liste de tous les restaurants

Récupérer une liste des restaurants qui livrent

Récupérer une liste de restaurants proches

Récupérer un restaurant

Orders

### Récupérer la liste de tous les restaurants livrant une zone

Pour récupérer la liste de tous les restaurants partenaires qui livrent une zone donnée, il suffit de préciser en paramètre de l'URI ci-dessous le code insee de l'adresse à livrer. Ainsi cette requête renverra les restaurants qui livrent dans la zone précisée. Vous pouvez trier les résultats grâce à des paramètres facultatifs.

GET

... /restaurants/delivery/insee={codeINSEE}&{params}

### Liste des paramètres

Paramètres d'url

- **insee**: code INSEE de la zone de livraison souhaitée
- **dates**: Permet de préciser les dates desquelles on souhaite avoir les horaires des restaurants, au format Y-m-d séparées par des virgules (ex: 2018-03-15,2018-03-16...) Facultatif
- **partnership**: Permet un tri les résultats suivant les valeurs suivantes : Facultatif
  - **all**: valeur par défaut : ressort tous les restaurants
  - **none**: ressort uniquement les restaurants auxquels vous n'avez pas encore proposé de partenariat
  - **waiting**: ressort uniquement les restaurants qui sont en attente de confirmation du contrat
  - **refused**: ressort uniquement les restaurants ayant refusé vos propositions de partenariat
  - **accepted**: ressort uniquement les restaurants ayant accepté vos propositions de partenariat

*Pas de corps pour cette requête*

*Exemple de réponse :*

JSON	XML	HTML	YML
<pre>[   {     "id": 1,     "name": "Le Port de Lagrange",     "adressLine1": "21 rue des Mimosas",     "adressLine2": null,     "countryCode": "33700",     "city": "Bordeaux",     "region": null,     "state": null   } ]</pre>			