OC Pizza

Dossier de conception technique

Version 1.0

Auteur
RACHIDI Raoof
Analyste/programmeur

RCD Informatique
139 rue baraban 69003 Lyon – 06.80.55.10.44 – raoof976@gmail.com

S.A.R.L. au capital de 1 000,00 € enregistrée au RCS de Xxxx – SIREN 999 999 999 – Code APE : 6202A

Table des matières

1 - Versions

Auteur	Date	Description	Version
Rachidi Raoof	12/04/19	Création du document	01/01/00

2 - Introduction

2.1 - Objet du document

Le présent document constitue le dossier de conception technique de l'application OC Pizza.

Ce document a pour but d'analyser la demande du client OC Pizza afin de :

- modéliser les objets du domaine fonctionnel ;
- identifier les différents éléments composant le système à mettre en place et leurs interactions ;
- décrire le déploiement des différents composants envisagés ,
- élaborer le schéma de la base de données.

2.2 - Besoin du client

2.2.1 - Contexte

« OC Pizza » est un jeune groupe de pizzeria en plein essor et spécialisé dans les pizzas livrées ou à emporter. Il compte déjà 5 points de vente et prévoit d'en ouvrir au moins 3 de plus d'ici la fin de l'année.

2.2.2 - Enjeux et objectifs

Un des responsables du groupe a pris contact avec vous afin de mettre en place un système informatique, déployé dans toutes ses pizzerias et qui lui permettrait notamment :

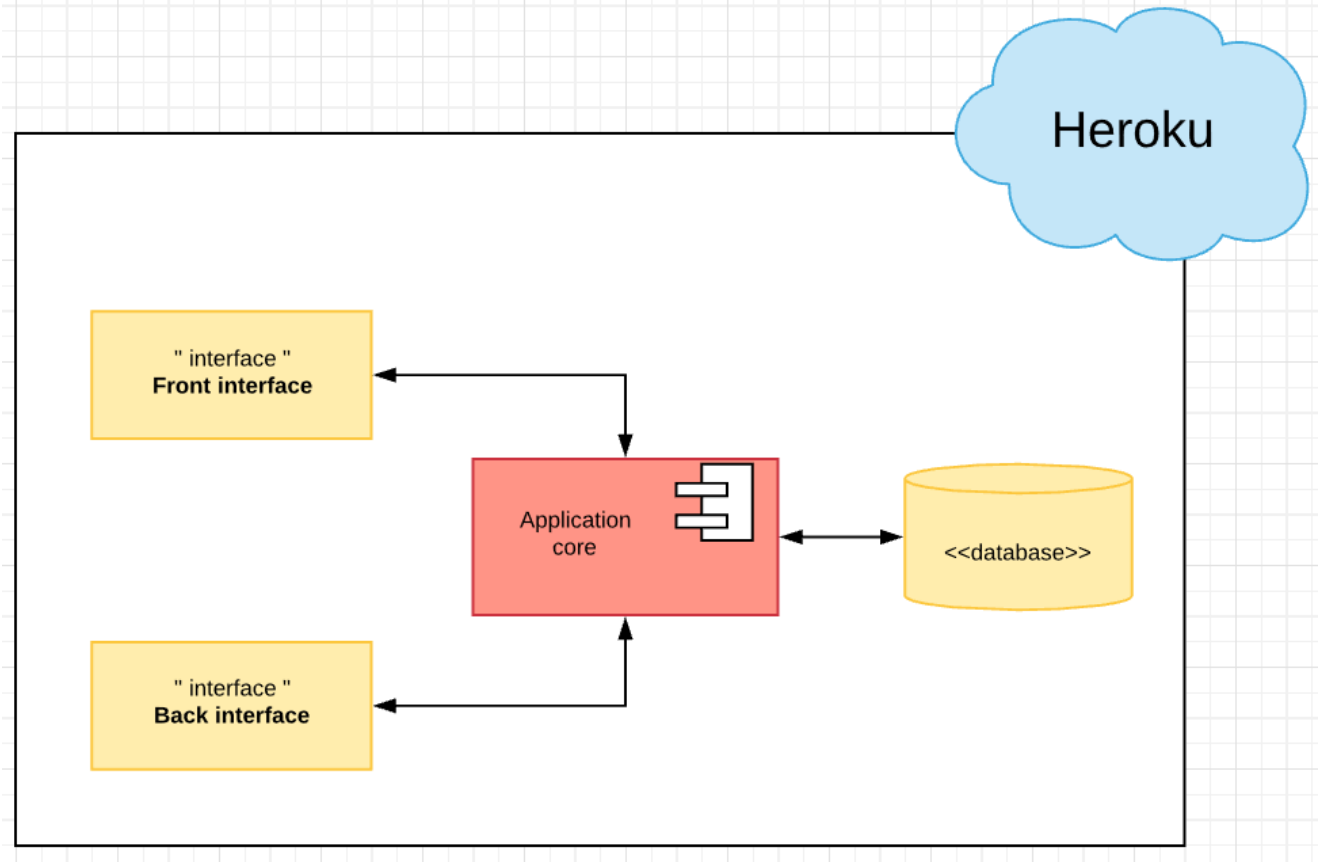
- d'être plus efficace dans la gestion des commandes, de leur réception à leur livraison en passant par leur préparation ;
- de suivre en temps réel les commandes passées et en préparation ;
- de suivre en temps réel le stock d'ingrédients restants pour savoir quelles pizzas sont encore réalisables ;
- de proposer un site Internet pour que les clients puissent :
    - passer leurs commandes, en plus de la prise de commande par téléphone ou sur place
    - payer en ligne leur commande s'ils le souhaitent – sinon, ils paieront directement à la livraison
    - modifier ou annuler leur commande tant que celle-ci n'a pas été préparée
- de proposer un aide mémoire aux pizzaiolos indiquant la recette de chaque pizza
- d'informer ou notifier les clients sur l'état de leur commande.

3 - Architecture Technique

3.1 - Généralités

L'application :

- sera composée de 2 interfaces différentes, une interface *front* (client) et une interface *back* (employé) ;
- se basera sur un socle technique et une base de données commune ;
- sera hébargé à l'aide d'une solution cloud flexible de type Heroku.



Avantages/inconvénients

L'avanage d'une telle solution est séparer les interfaces du coeur de l'application et ainsi en faire une application modulable et extensible.

De plus, cela en fait une solution bien plus simple à maintenir dans le temps avec les évolutions techniques futures.

Il y a aussi un avantage non négligeable à déployer l'application sur un cloud, car cela ôte la contrainte budgétaire et temporaire d'installation et gestion de serveurs.

L'inconvénient étant de dépendre alors d'un prestataire tiers et de son bon vouloir.

3.2 - Pile logicielle

3.2.1 - Base de données

La base de données est une base **PostgreSQL**, comme **recommandé par Heroku**.
Son collationnement et son encodage sont en **UTF-8**.

3.2.2 - Paquet coeur

Serveur :

Langage de programmation : Python 3.8.0
ORM : SQLAlchemy 1.3.11

3.2.3 - Interfaces

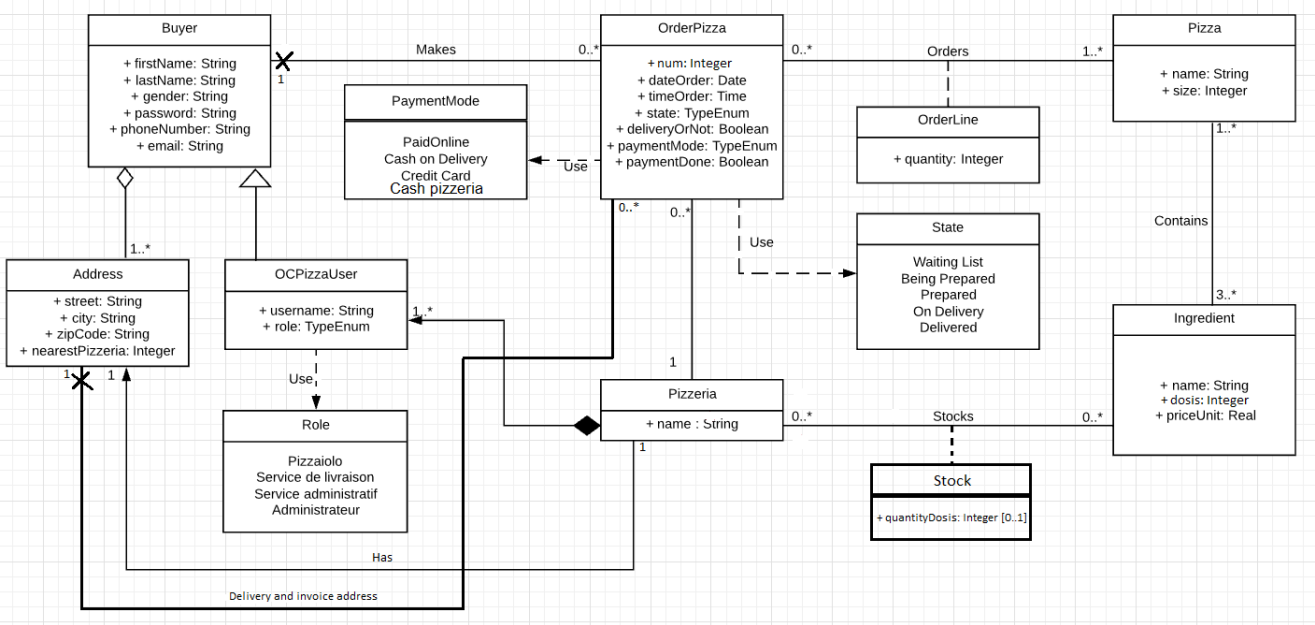
Serveur :

Langage de programmation : Python 3.8.0
Framework web + ORM : Django 2.2.7
Serveur web : Gunicorn 20.0.4

Client :

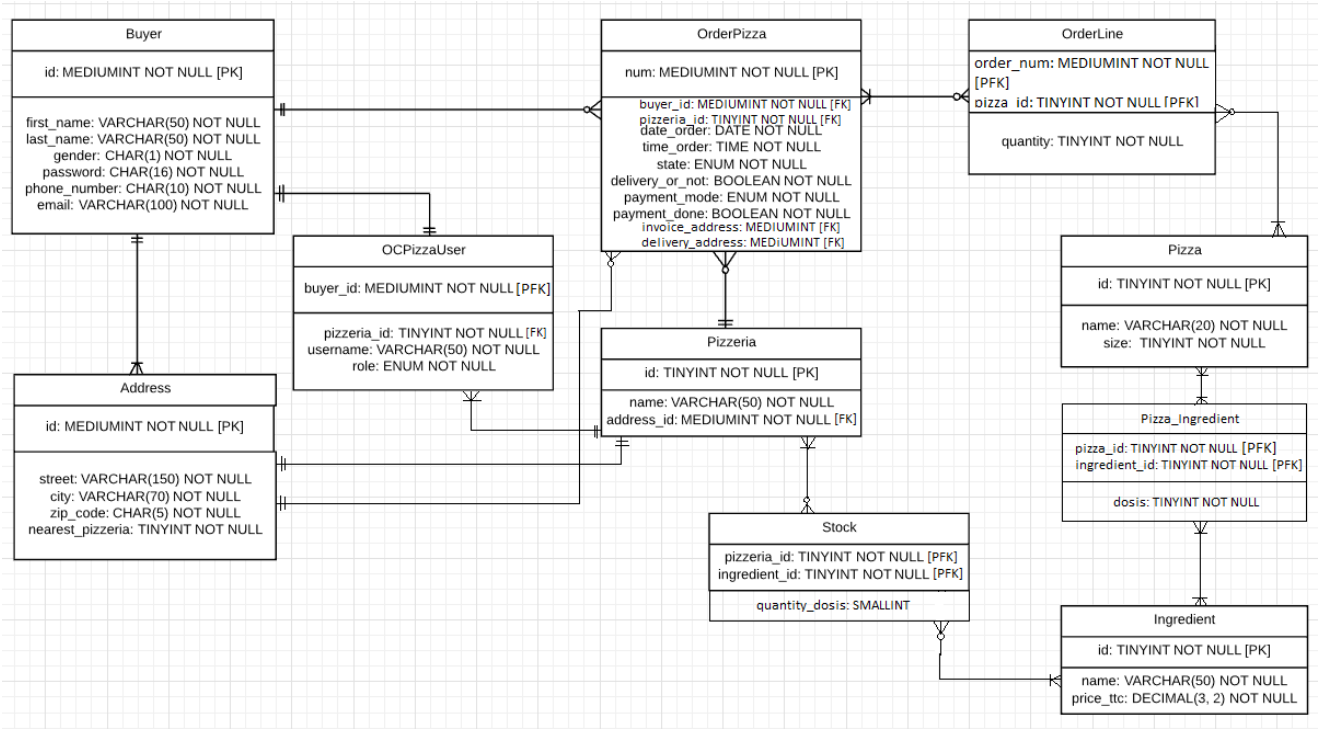
Framework d'interface : Bootstrap 4.3.1
Framework web : VueJS 2.6.10

4 - Architecture de l'application



Pour réaliser la strucure de la base de données, nous avons tout d'abord réalisé le diagramme de classes. Ce diagramme nous a permis de définir comment l'application sera structurée et réaliser les tables de la base de données en conséquence.

5 - Architecture de base de données



La base de données est composée des tables présentes sur l'image ci-dessus.
Sur les pages suivantes, vous retrouverez un aperçu de chaque table et une description des points importants ainsi qu'une explication des clés étrangères.

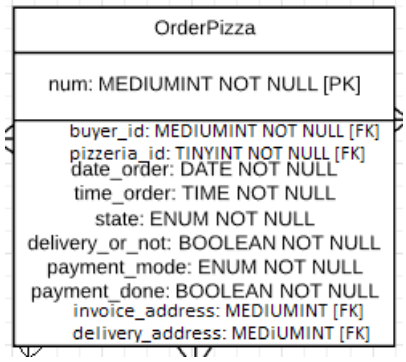
Table “ OrderPizza ”	
	**OrderPizza :** Regroupe les attributs nécessaires pour une commande de pizza. *FK : buyer_id → permet de rattacher une commande à un utilisateur* *FK : pizzeria_id → permet de rattacher une commande à une pizzeria*

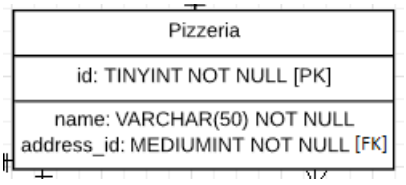
Table “ Pizzeria ”	
	**Pizzeria :** Stocke les différents restaurants, leur nom et leurs coordonnées de contact. *FK : address_id → permet d'avoir l'adresse de la pizzeria*

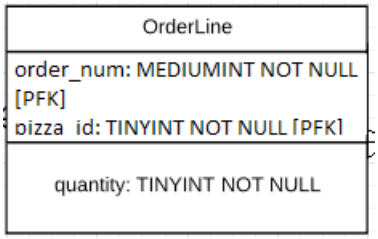
Table “ OrderLine ”	
	**OrderLine :** Table de jonction entre **OrderPizza** et **Pizza**. *PFK : order_num → permet d'identifier une commande* *PFK : pizza_id → permet de rattacher une pizza à un numéro de commande*

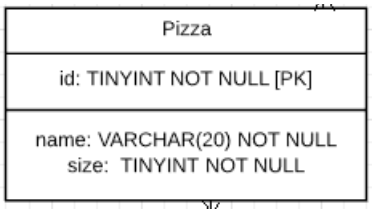
Table “ Pizza ”	
	**Pizza :** Stocke toutes les pizzas de tous les restaurants. Une description et une image sont optionnels.

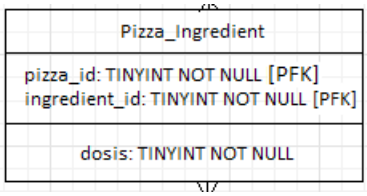
Table “ Pizza_Ingredient ”	
	**Pizza_Ingredient :** Table de jonction entre **Pizza** et **Ingredient**. Définit la quantité d'ingrédient nécessaire à la réalisation d'un produit (plat). *PFK : pizza_id → identifie une pizza* *PFK : ingredient_id → identifie un ingrédient*

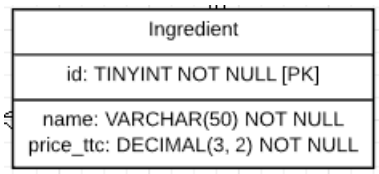
Table “ Ingredient ”	
	**Ingredient :** Stocke tous les ingrédients de tous les restaurants.

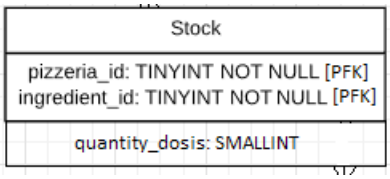
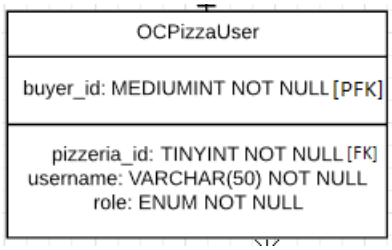
Table “ Stock ”	
	**Stock :** Permet la gestion des stocks des aliments de chaque restaurant. *PFK : pizzeria_id → permet d'identifier la pizzeria concernée* *PFK : ingredient_id → permet d'identifier l'ingrédient en stock*

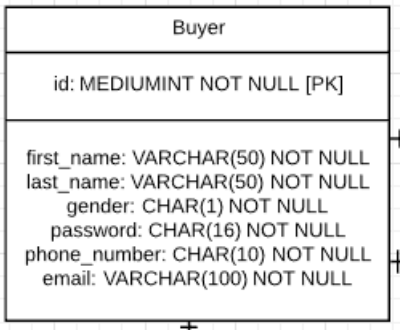
Table “ OCPizzaUser ”



OCPizzaUser :
Permet de définir le rôle d'un utilisateur.

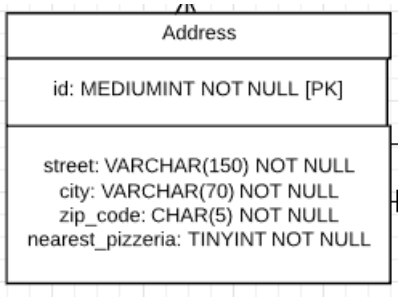
pizzeria_id : → permet de rattacher un employé à un restaurant

Table “ Buyer ”



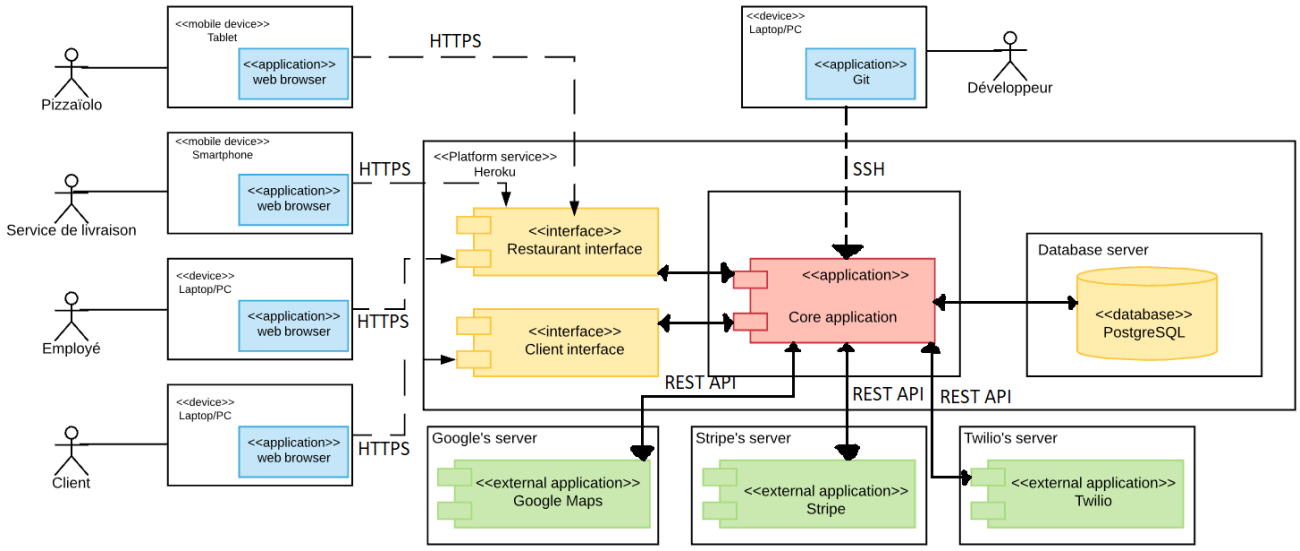
Buyer :
Permet de stocker tous les clients de l'application.

Table “ Address ”



Address :
Stocke les adresses des utilisateurs (clients/employés).

6 - Architecture de déploiement



Le diagramme de déploiement nous permet de comprendre comment sera déployée l'application et par quel moyen chaque acteur devrait y accéder.

L'application étant déployée par Heroku, **les développeurs ont simplement besoin d'un terminal** de commande pour déployer celle-ci en production via **git**.

6.1 - Interfaces

Les utilisateurs (employés/clients) accèdent à l'application via leur terminal favori ou le plus indiqué et un navigateur web installé sur celui-ci. La communication se fait via le protocole HTTPS.

Les clients et les employés ont accès à une interface différente.

6.2 - Terminaux recommandés

Sur le diagramme de déploiement est indiqué le terminal recommandé pour le pizzaïolo et le service de livraison.

En effet, il serait plus pratique en cuisine d'avoir une tablette qui prend peu de place pour le pizzaïolo. Le service de livraison lui utiliserait un smartphone afin d'avoir une connexion 3G/4G et récupérer en temps réel le chemin le plus rapide/pratique pour rejoindre l'adresse du client et le livrer au plus tôt.

6.3 - Application

Le paquet logiciel Python (core) regroupe l'ensemble des fonctionnalités qui effectuent la liaison entre les interfaces et les serveurs externes des API, ainsi que la base de données de l'application.

L'intérêt de cette démarche est de réduire le temps et le coût de développement, d'améliorer la sécurité globale du logiciel et permettre le développement d'interfaces supplémentaires, si nécessaires, en regroupant toutes les fonctionnalités qui peuvent l'être et ainsi obtenir une **application modulable**.

Le coeur permet de faire le lien entre la base de données, les interfaces et les API externes.

6.4 - Serveur de l'application

L'application sera déployée sur Heroku, une plateforme cloud.

Cela permettra à OC Pizza de pouvoir payer la solution en fonction de son utilisation et ainsi payer un tarif en fonction de l'affluence sur la plateforme.

6.5 - Serveur de base de données

La base de données PostgreSQL sera déployée sur le serveur Heroku via Heroku Postgres. Le choix d'une base de données PostgreSQL est préférable car bien intégrée avec Heroku.

7 - Points particuliers

7.1 - Environnement de développement

7.1.1 - Environnement virtuel

Il est toujours **fortement recommandé** d'utiliser un environnement virtuel lors du développement d'une application Python (via virtualenv, conda, ou pipenv). Cela permet en effet d'installer des dépendances dans un environnement qui n'est pas l'environnement principal de la machine, et de **gérer des dépendances** en fonction d'un projet particulier. Cela permet aussi **d'éviter beaucoup de conflits de versions** et d'éviter les mises à jour non prévues.

Une fois l'environnement virtuel installé, il suffit d'installer les dépendances via pip.

8 - Acteurs externes

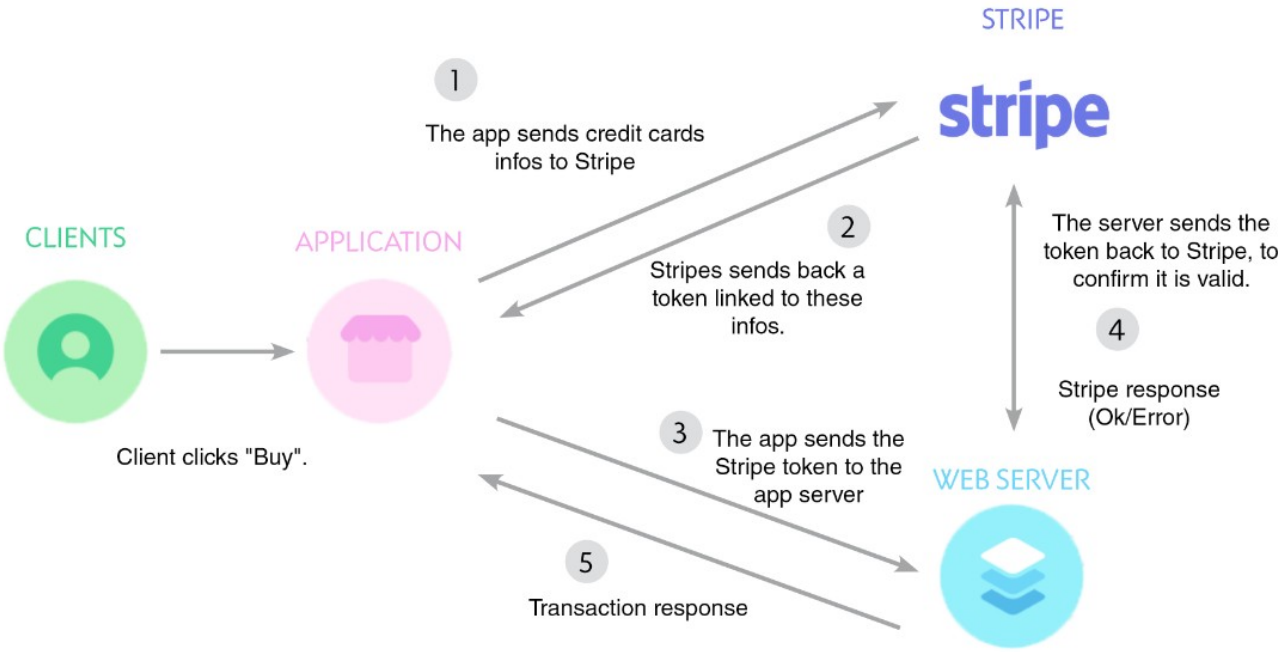
8.1 - Paiements : Stripe

Gestion des paiements via l'application web.

Très simple à mettre en place, sécurisé et très répandu sur les sites de e-commerce actuels (Deliveroo, Lyft, Ulule...). Documentation très fournie.

Le module **dj-stripe** sera utilisé pour intégrer Stripe à l'application Django crée.

Voici un petit schéma explicatif de comment Stripe fonctionne (inspiré d'un diagramme fourni par Stripe).



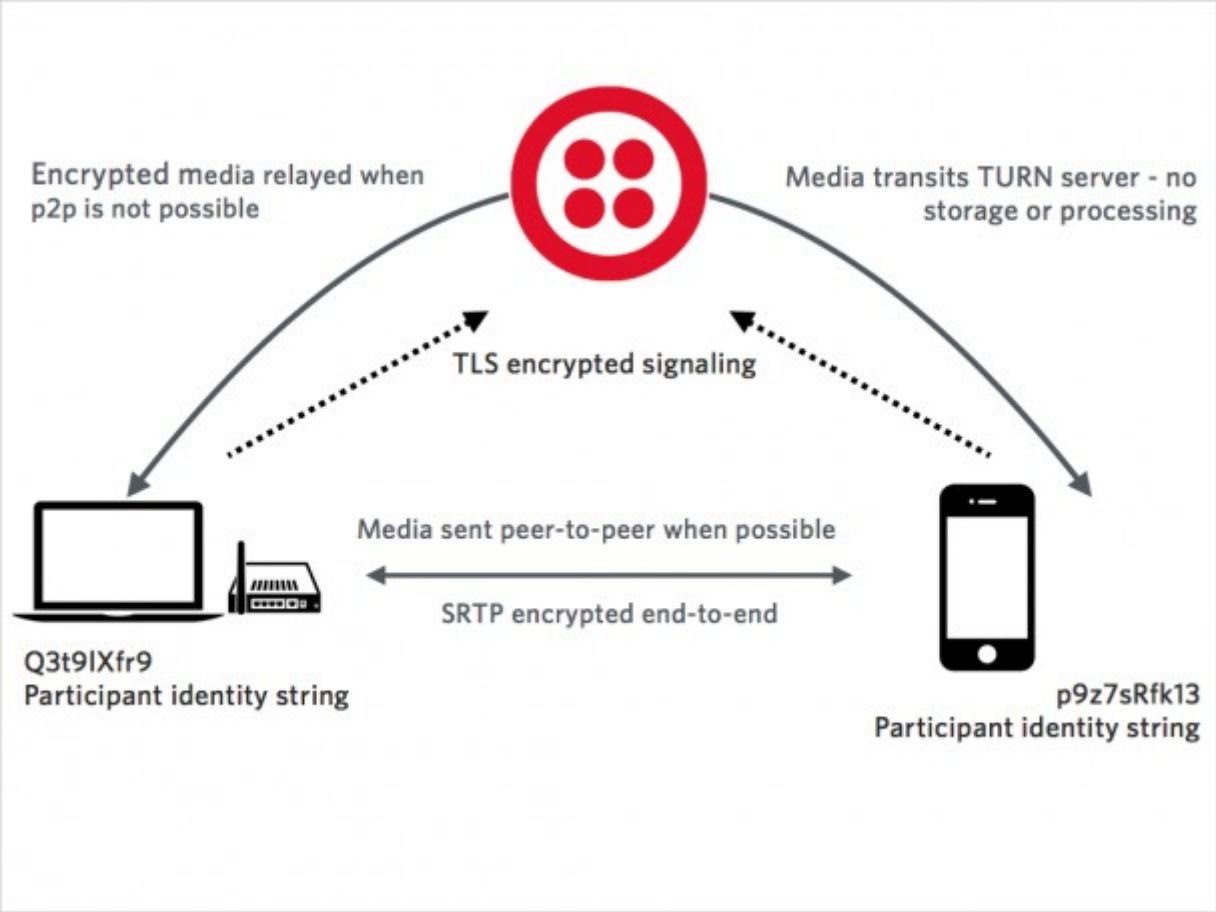
8.2 - Notifications : Twilio

Permet de notifier l'utilisateur par SMS mais aussi directement dans le navigateur (push). La mise en place est assez simple et la documentation est fournie.

Twilio fourni une librairie Python pour intégration.

Le module **django-twilio** sera utilisé pour intégrer Twilio à l'application Django créée.

Voici un simple schéma permettant de comprendre (très) rapidement comment Twilio fonctionnerait avec notre web application OC Pizza.



La base de données communiquerait au coeur de l'application les informations client nécessaire (le numéro de téléphone), la commande associée et son statut actuel.

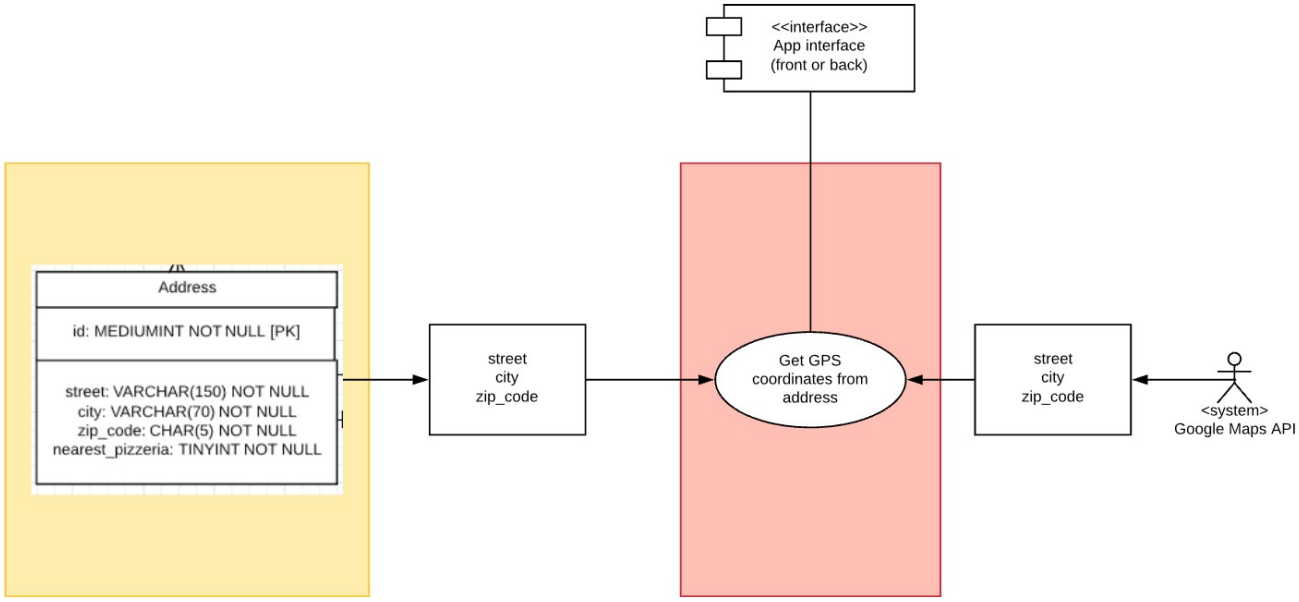
Ensuite, suivant les paramètres souhaités par OC Pizza, une notification SMS serait envoyée au client pour un /des statuts particuliers.

8.3 - Maps : Google Maps

- Affiche la position des restaurants pour l'utilisateur ;
- Permet aux livreurs de se rendre efficacement chez le client grâce aux informations sur le trafic ou les travaux en cours.

Il existe une librairie créée par la communauté Python pour utiliser facilement Google Maps au sein d'une application Python/Django. (https://github.com/googlemaps/google-maps-services-python)

L'API Google Maps permettra de transformer les adresses des utilisateurs/restaurants en coordonnées GPS.


<<interface>>
App interface
(front or back)
Address
id: MEDIUMINT NOT NULL [PK]
street: VARCHAR(150) NOT NULL
city: VARCHAR(70) NOT NULL
zip_code: CHAR(5) NOT NULL
nearest_pizzeria: TINYINT NOT NULL
street
city
zip_code
Get GPS
coordinates from
address
street
city
zip_code
<system>
Google Maps API