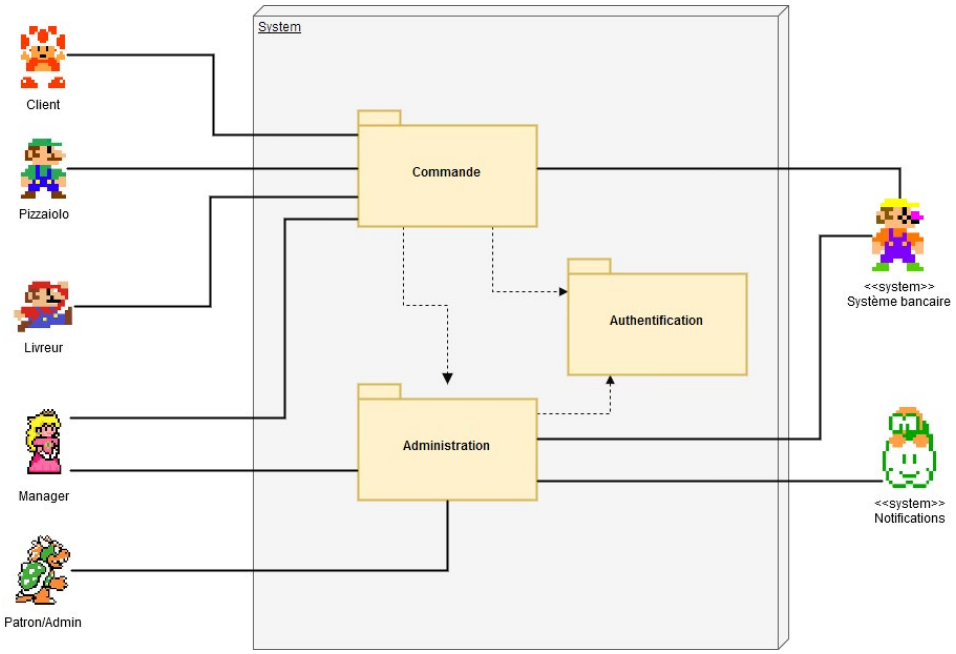
PARCOURS OPENCLASSROOMS DÉVELOPPEUR D'APPLICATIONS PYTHON

QUENTIN LATHIÈRE

Projet 6 : Concevez la solution technique d'un système de gestion de pizzeria

https://github.com/Synkied/OC_Projet-6



Le présent document résume la réalisation du 6e projet (Concevez la solution technique d'un système de gestion de pizzeria) dans le cadre de mon parcours Développeur d'application - Python.

Versions

Auteur	Date	Description	Version
Quentin	10/01/2018	Création du document	1
Quentin	12/01/2018	Ajout des cas d'utilisation	2
Quentin	16/01/2018	Ajout description BDD	3

Table des matières

I. Objet du document

Objet

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

II. Besoin du client

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.

Enjeux et objectifs

Un des responsables du groupe a pris contact avec nous 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.

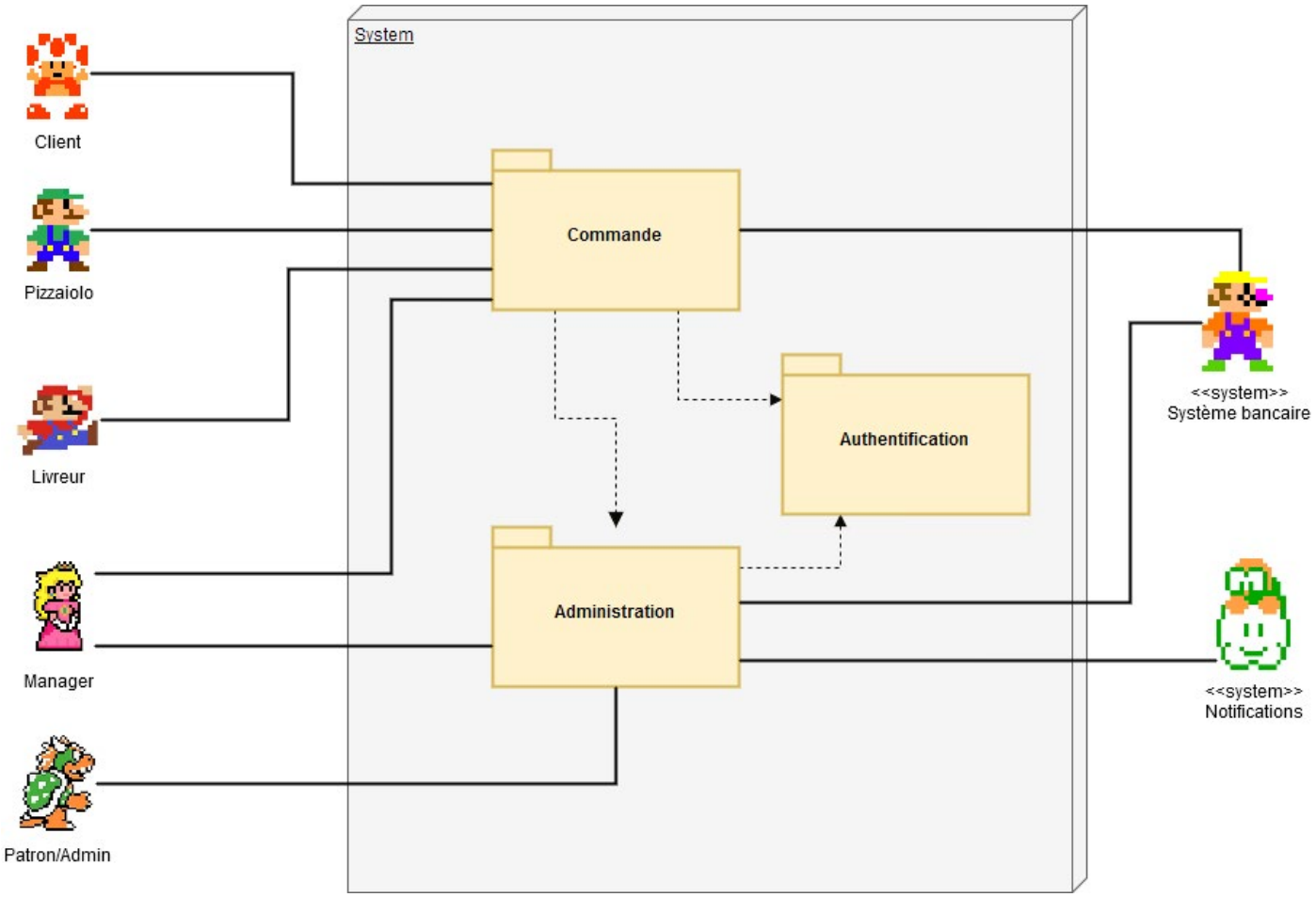
Prospection

Le client a déjà fait une petite prospection et les logiciels existants qu'il a pu trouver ne lui conviennent pas.

III. Architecture technique

1. Composants généraux

Le diagramme de paquetage suivant représente les paquetages principaux de l'application à développer. (Des personnages du jeu vidéo Mario ont été utilisés pour la présentation au client et simplifier la démarche, ils sont donc présents sur ce diagramme.)



a. Paquetage de commande

Le **paquetage de commande** permet au client de consulter les produits.
Il peut ensuite passer une commande, en consulter son état et régler le paiement de celle-ci.
Le client peut aussi modifier sa commande si elle n'a pas été préparée. Il devra alors payer le complément ou se faire rembourser la différence auprès du système de paiement.

b. Paquetage d'administration

Le **paquetage d'administration** est utile à plusieurs niveaux.
Il permet, suivant les accès de chaque acteur, de :

- Gérer les stocks des restaurants ;
- Gérer les produits présents dans les restaurants (en plus des pizzas, les entrées possibles, ainsi que les desserts et boissons) ;
- Gérer les commandes et notifier l'état de la commande au client.

Seul l'administrateur a accès à la gestion de ces trois éléments pour tous les restaurants.
Les autres acteurs (manager/employés), le pizzaiolo et le livreur, n'ont accès qu'aux informations du restaurant dont ils dépendent.

c. Paquetage d'authentification

Le **paquetage d'authentification** permet de gérer toute la partie enregistrement/connexion d'un utilisateur.
Un utilisateur peut être un client ou tout employé des restaurants.

Il est possible d'enregistrer des nouveaux utilisateurs ou de permettre à un utilisateur déjà enregistré de se connecter.

Un utilisateur déjà enregistré peut aussi faire une demande de réinitialisation de mot de passe s'il a oublié celui-ci.

Une vérification dynamique des différents éléments de compte se fait lors de la création ou la modification de chaque élément de compte (mot de passe, adresse de livraison, nom, prénom, et nom d'utilisateur).

2. Application web

Diagramme de composants
La pile logicielle est la suivante :

- Application Django 2.0
- Serveur d'application Heroku
- Base de données PostgreSQL
- Interpréteur de scripts : Python 3

a. Composants

Internes :
Application : Django
L'application créée est divisée en deux parties distinctes :
Une partie *front* pour les clients et une partie *back* pour les employés.

Front :
La partie client est simple et sobre. Permet au client de choisir des produits dans des catégories, de créer un compte. Le client peut créer un panier et payer directement sur l'application web.

Back
La partie employée est plus complexe. Elle permet de passer des commandes mais aussi de les gérer (modification/suppression).
Une partie stock permet de contrôler les stocks et les réapprovisionner.
Il est aussi possible pour un administrateur avec les droits d'ajouter, modifier ou supprimer des produits/catégories du catalogue.
La partie employée permet aussi de gérer les utilisateurs et modifier leurs informations.
Les accès sont gérés grâce aux rôles des utilisateurs.

Externes :

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, Instaart, Slack, Ulule...). Documentation très fournie.

EXPLICITER

Notifications : twilio

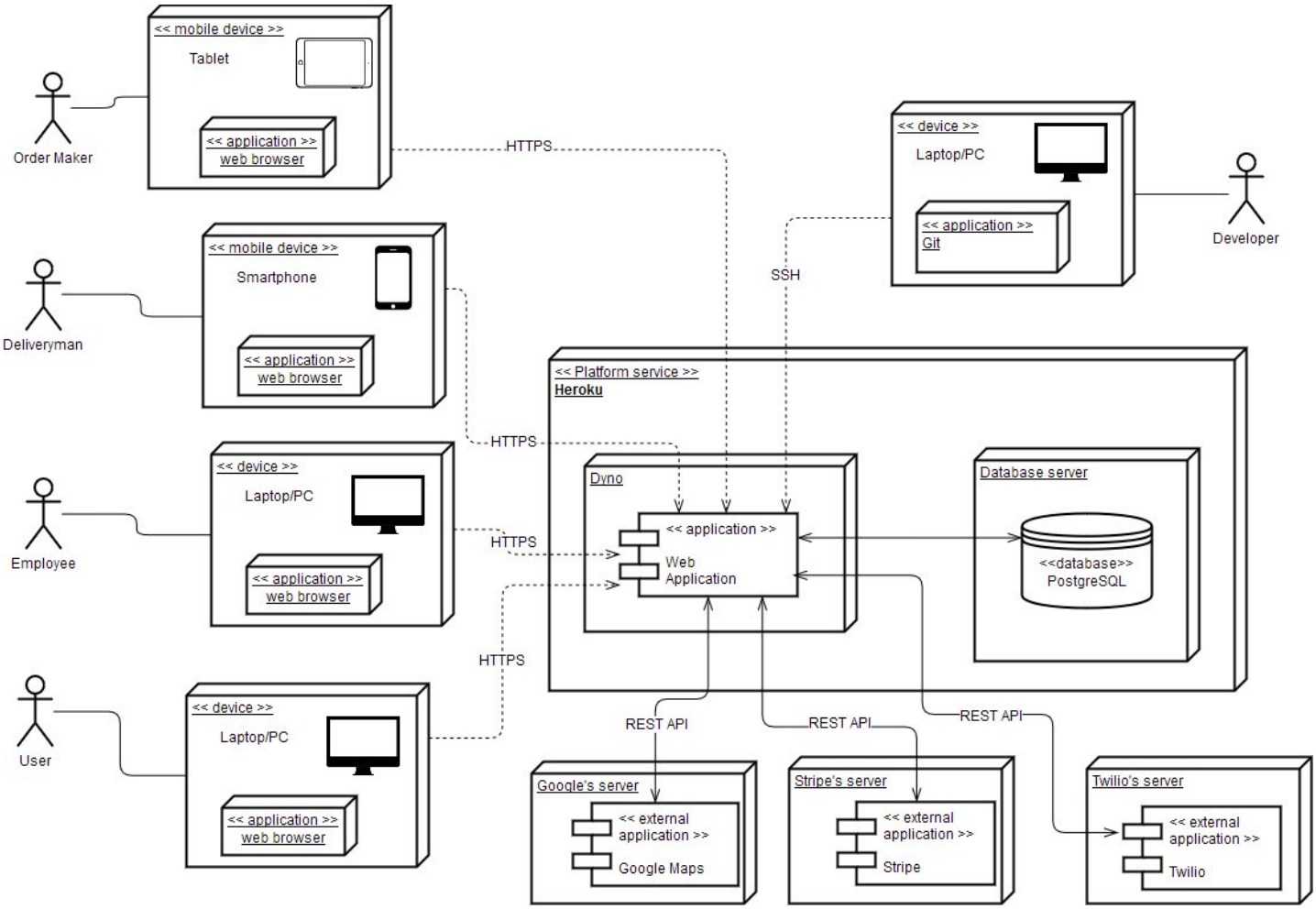
Permet de notifier l'utilisateur par SMS mais aussi directement dans le navigateur (push).
Scalable, mise en place assez simple et documentation fournie.
Twilio fourni une librairie Python pour intégration.
EXPLICITER

Maps : Google Maps

- Affiche la position des restaurants pour l'utilisateur
- Permet au livreur 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)
EXPLICITER

IV. 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 sur Heroku, les développeurs ont simplement besoin d'un terminal de commande pour déployer celle-ci en production.

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

Terminaux recommandés

Sur le diagramme de déploiement est indiqué le terminal recommandé pour le pizzaiolo (order maker) et le livreur.
En effet, il serait plus pratique en cuisine d'avoir une tablette qui prend peu de place pour le pizzaiolo. Le livreur lui utiliserait un smartphone afin d'avoir une connexion 3G/4G/GPS et récupérer en temps réel le chemin le plus pratique/rapide pour rejoindre l'adresse du client et le livrer au plus tôt.

1. Serveur de l'application

L'application sera déployée sur Heroku.

2. Serveur de base de données

La base de donnée PostgreSQL sera déployée sur le serveur Heroku.

V. Architecture logicielle

1. Principes généraux

Les sources et versions du projet sont gérées par Git.
Une fois une version validée en locale, elle est déployée sur Heroku.
Les dépendances sont gérées avec Distutils via un fichier setup.py.

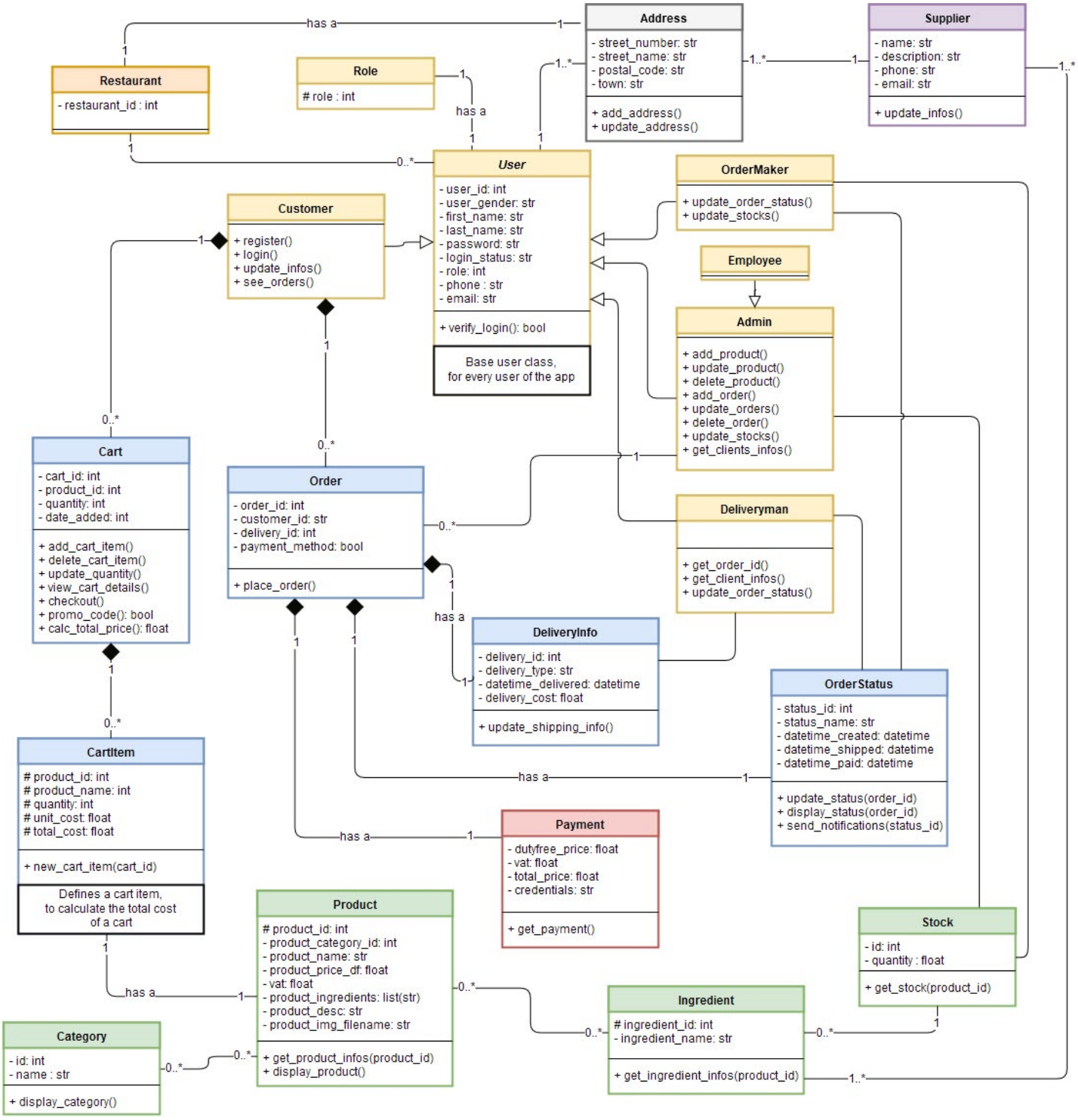
a. Les couches

L'architecture applicative est la suivante :
Django : modèle vue template, côté application
PostgreSQL : modèle, côté base de données

b. Les modules

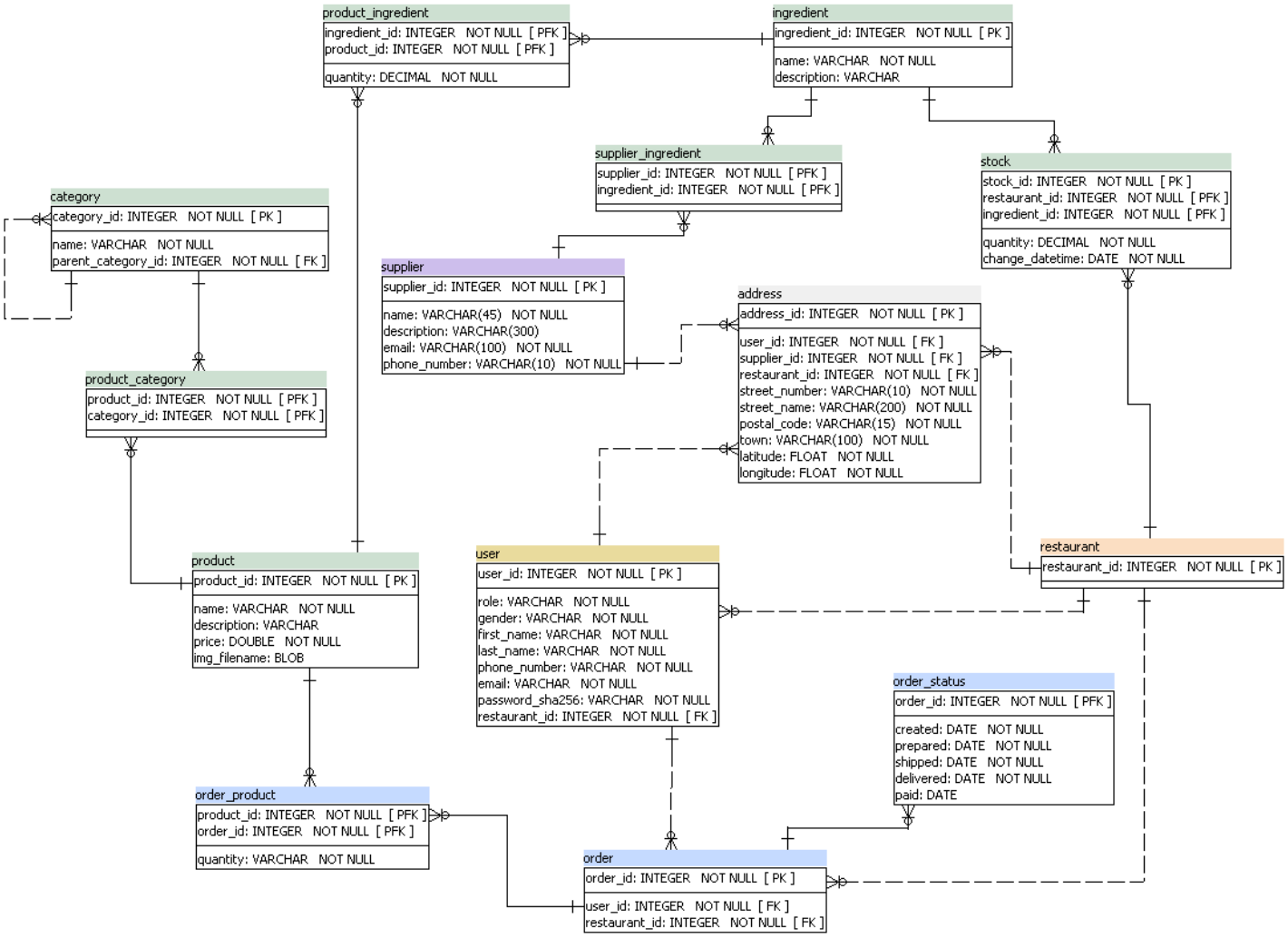
django-redis : pour la gestion du cache du navigateur web de l'utilisateur.
django-simple-captcha : afin de vérifier que l'utilisateur n'est pas un robot.

VI. Architecture de l'application



Pour réaliser la structure de 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 de réaliser les tables de la base de données en conséquence.

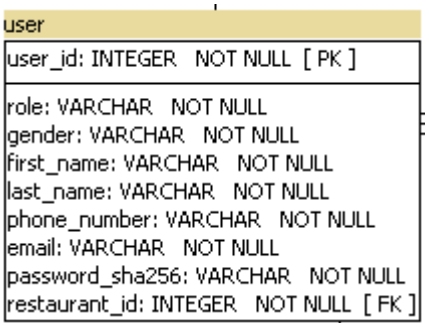
VII. Architecture de base de données



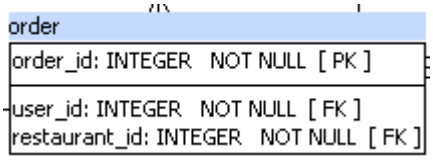
La base de données est composée des tables présentent sur l'image ci-dessus.

User :

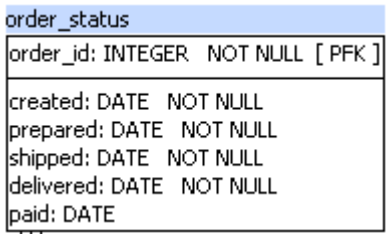
Permet de stocker tous les utilisateurs de l'application.
role : l'utilisateur a un rôle qui permet de définir ses accès au sein de l'application.
password_sha256 : un hash du mot de passe est stocké en base, pas le mot de passe directement.
FK : restaurant_id -> permet de rattacher un employé ou un client à un restaurant.



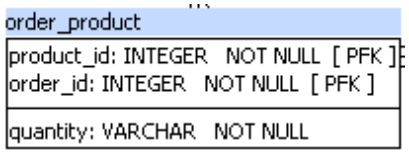
Order :
Permet d'enregistrer toutes les commandes passées par un client ou un employé.
FK : restaurant_id -> permet de rattacher une commande à un restaurant.
FK : user_id -> permet de rattacher une commande à un utilisateur (client/employé).



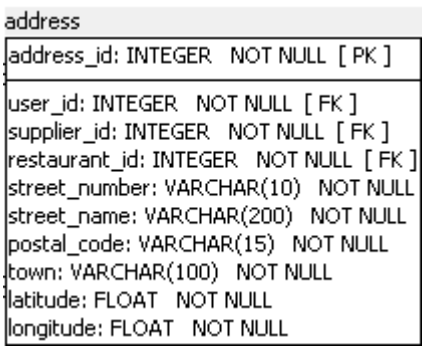
Order_status :
Permet de stocker le statut d'une commande et savoir a tout instant où celle-ci en est, notamment pour notifier le client et les employés du restaurant.
Le statut est un booléen.
PFK : order_id -> rattache une commande à son statut.



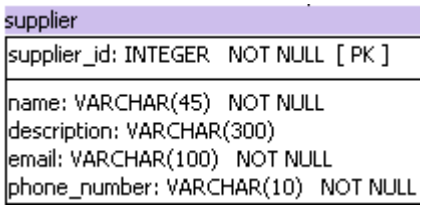
Order_product :
Table de jonction entre **order** et **product**.
Permet de savoir quelle commande contient quel produit et quelle quantité de ce produit.
PFK : product_id -> permet d'identifier un produit dans une commande.
PFK : order_id -> permet d'identifier la commande.



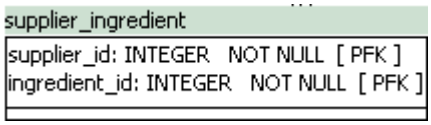
Address :
Stock les adresses des utilisateurs et fournisseurs.
FK : user_id -> permet d'identifier si l'adresse est celle d'un utilisateur et lequel.
FK : supplier_id -> permet d'identifier si l'adresse est celle d'un fournisseur et lequel.



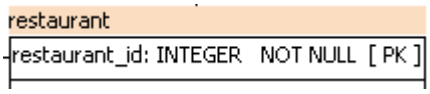
Supplier :
Stock tous les fournisseurs de tous les restaurants.



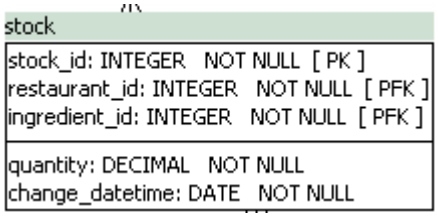
Supplier_ingredient :
Table de jonction entre **supplier** et **ingredient**.
Possibilité d'avoir plusieurs fournisseurs qui fournisseurs plusieurs ingrédients.
PFK : supplier_id -> identifie un fournisseur.
PFK : ingredient_id -> identifie un ingrédient.



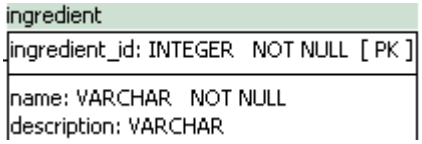
Restaurant :
Stock les différents restaurants et leur adresse physique (position géographique).



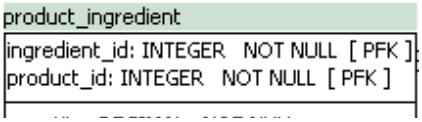
Stock :
Permet la gestion des stocks des aliments de chaque restaurant.
Un timestamp permet de savoir quand le stock a été changé (ajout/suppression).
FK : ingredient_id -> permet d'identifier l'ingrédient en stock.
FK : restaurant_id -> permet d'identifier le restaurant auquel est attaché l'ingrédient.



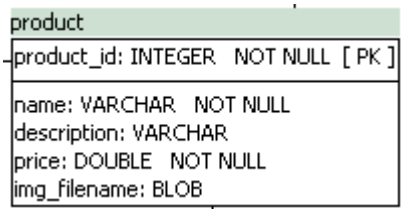
Ingredient :
Stock tous les ingrédients de tous les restaurants.
FK : suppliers_id -> permet d'identifier le fournisseur habituel de l'ingrédient en question.



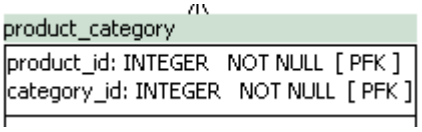
Product_ingredient :
Table de jonction entre **ingredient** et **product**.
Défini la quantité d'ingrédient nécessaire à la réalisation d'un produit.
PFK : ingredient_id -> identifie un ingrédient.
PFK : product_id -> identifie un produit.



Product :
Stock tous les produits de tous les restaurants.
Une description et une image sont optionnels.



Product_category :
Table de jonction entre **product** et **category**.
Permet d'identifier plusieurs produits dans plusieurs catégories.
PFK : product_id -> identifie un produit.
PFK : category_id -> identifie une catégorie.



Category :
Stock les différentes catégories dans lesquelles peuvent être classés les produits.
Une catégorie peut avoir une ou plusieurs catégories parentes.