# {{OC Pizza}}

# {{OC Pizza-Manager}}

Dossier de conception Technique

Version {{1.0.0}}

**Auteur**
{{Mr Cottenceau Thomas}}
*{{Développeur Full Stack Python}}*

# TABLE DES MATIÈRES


**1 - Versions** ..... **3**
**2 - Introduction** ..... **4**
2.1 - Objet du document ..... 4
2.2 - Références ..... 4
**5 - Architecture Technique** ..... **5**
5.1 - Composants généraux ..... 5
*5.1.1 - Languages de programmation* ..... *5*
5.2 - Application Web ..... 6
*5.2.1 - Github* ..... *6*
*5.2.2 - Django* ..... *6*
*5.2.3 - Heroku* ..... *6*
*5.2.4 - PostgrèsSQL* ..... *6*
**6 - Architecture de Déploiement** ..... **7**
**7 - Architecture logicielle** ..... **8**
7.1 - Principes généraux ..... 8
*7.1.1 - Les couches* ..... *8*
*7.1.2 - Structure des sources* ..... *9*
7.2 - Application Web ..... 10
**8 - Points particuliers** ..... **11**
8.1 - Gestion des logs ..... 11
8.2 - Fichiers de configuration ..... 11
*8.2.1 - Application web* ..... *11*
*8.2.2 - Applications Django* ..... *15*
8.3 - Ressources ..... 16
8.4 - Environnement de développement ..... 17
8.5 - Les Tests ..... 17
8.6 - Procédure de packaging / livraison ..... 17

# 1 - VERSIONS

| Auteur | Date | Description | Version |
|---|---|---|---|
| Cottenceau Thomas | 02/03/2021 | Création du document | 1.0.0 |
| | | | |
| | | | |
| | | | |

# 2 - INTRODUCTION

## 2.1 - Objet du document

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

“OC Pizza-Manager”.

Il détaille dans l’ensemble les caractéristiques techniques générales de l’application.

## 2.2 - Références

Pour de plus amples informations, se référer également aux éléments suivants:

3 - **DCT – OC Pizza-Manager** : Dossier de conception fonctionnelle

4 - **DCT - OC Pizza-Manager** : Dossier d’exploitation

# 5 - Architecture Technique

## 5.1 - Composants généraux

### *5.1.1 - Languages de programmation*

#### *5.1.1.1 - Python*

Python est le langage d'aujourd'hui, et ce pour plusieurs raisons:

Sa simplicité; la prise en main rapide de ce language due à sa syntaxe simple fait qu'il est clair, épuré et donc facile à éxecuter.

Sa popularité; qui ne fait que croître depuis pas mal d'années en fait un des languages les plus populaires grace à une communauté énorme.

Sa polyvalence; on le se retrouve dans tout genre de projet, du Frontend à l'I.A, et ce grâce aux milliers de librairies dont il dispose.

#### *5.1.1.2 - HTML/CSS*

Il nous faudra réaliser nos templates et les styliser et pour ce faire, seuls ces deux languages en sont capables.

#### *5.1.1.3 - JQuery*

Comme dans toute application il est souvent necessaire de pouvoir faire de l'AJAX afin de render plus vivant l'interface utilisateur. Cette librairie Javascript en est capable. De pus JQuery est simple d'utilisation, plus besoin d'être un expert en Javascript.

#### *5.1.1.4 - SQL*

Nous l'utiliserons pour faire le lien entre le système de modèle gérer par l'ORM de Django et l'éxecution d'instruction en base de données. Nous allons avoir besoin de créer dans nos classes objet (Pizza, Client, Employé pour ne citer qu'eux) des méthodes utiles aux traitement, sauvegarde et modifications de données.

## 5.2 - Application Web

La pile logiciel est la suivante:

- Système de versioning du code: **Github**
- Framework: **Django**
- Serveur d'application: **Heroku**
- Base de données: **PostgrèsSQL**

### *5.2.1 - Github*

Nous utiliserons le système de versioning très connus Github afin de versionner simplement et éfficacement notre code. Ce système est simple à mettre en place et ne nécessite pas d'être un pro pour en appréhender les commandes de base utiles au commencement d'un tel projet. De plus Github facilite énormément le travail d'équipe.

### *5.2.2 - Django*

Ce framework opensource Python n'est plus à présenter, Instagram, Google, Spotify, Netflix, Uber, Dropbox ou encore Pinterest peuvent en témoigner. Simple à prendre en main, il couvre tout les cas d'utilisations nécessaires au fonctionnement d'une plateforme et propose une architecture de projet solide et éfficace dans son éxecution.

### *5.2.3 - Heroku*

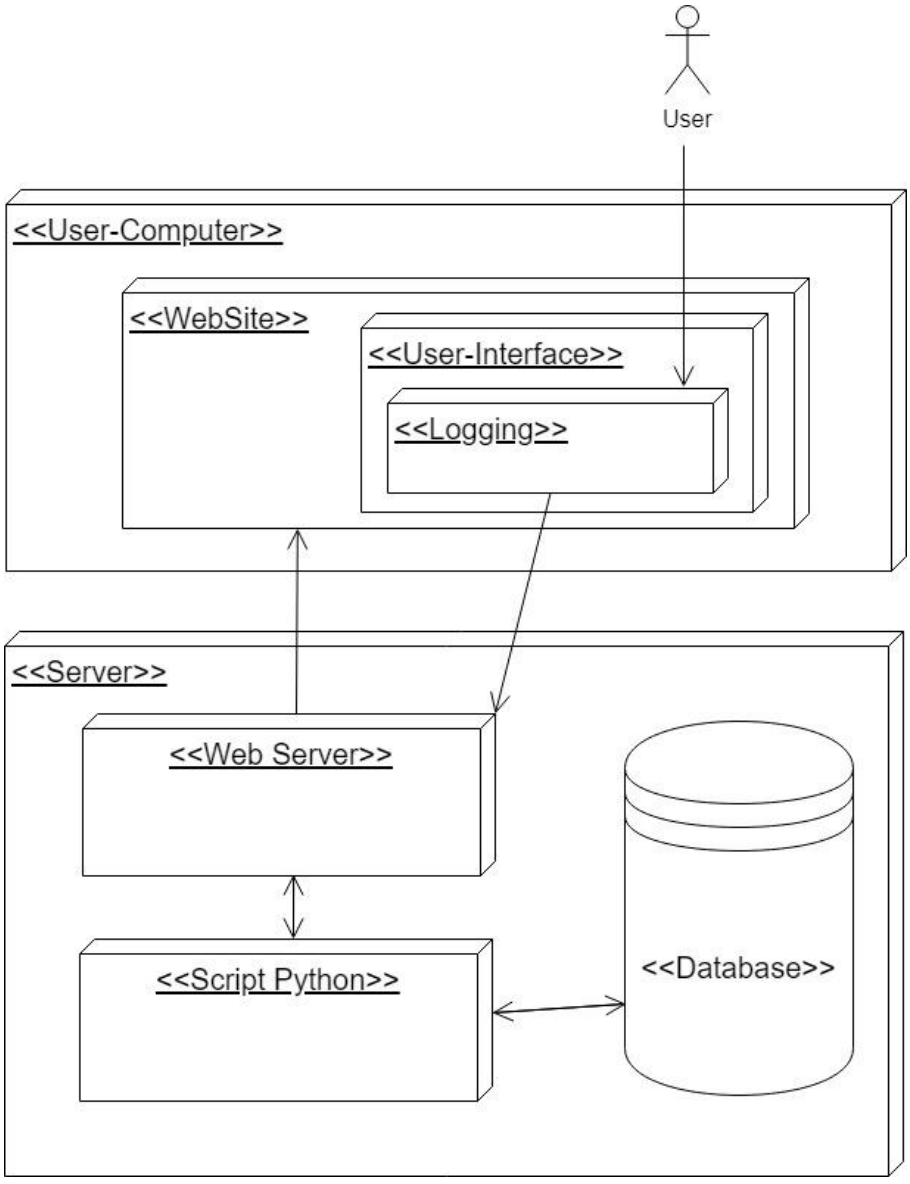
Notre application sera déployée sur le serveur Heroku car bon marché pour les possibiltés qu'il offre mais surtout car en terme de protection des données il est à la pointe. De plus il est compatible avec un grand nombre de language de programmation dont le Python et ne nécessite pas de mise en place particulière pour accueillir ce language. Enfin il offre la possibilité de travailler avec une base de données Postgrès ce qui dans notre cas colle parfaitement avec notre projet écrit en Python.

### *5.2.4 - PostgrèsSQL*

Postgrès est un système de gestion de base de données relationnelle ayant la particularité de pouvoir traiter un très grand nombre de données sans encombre. En outre il propose avec le concours de Python (orienté objet) une exploitation de modèle de données complexes. Dans notre cas, avec l'utilisation de l'ORM mis en place par Django Postgrès est donc LE système à utiliser. Nous utiliserons la dernière version 13.2. compatible sous Mac, Linux et Windows.

# 6 - Architecture de Déploiement

Diagramme UML de déploiement



# 7 - ARCHITECTURE LOGICIELLE

## 7.1 - Principes généraux

Comme expliqué précédemment les sources et versions du projet sont gérées par **Git**, les dépendances et le packaging par **Heroku.**

### *7.1.1 - Les couches*

L'architecture applicative du framework Django suit la logique MVT (modèle, vue et template). On distingue trois couches différentes.

Une couche **Model** : Le modèle interagit avec la base de données. Sa mission est de chercher dans la base de données les éléments correspondant à une requête et de renvoyer une réponse facilement exploitable par le programme. Les modèles s'appuient sur un ORM (*Object Relational Mapping*, ou Mapping objet-relationnel en français) qui traduit les résultats d'une requête SQL en objet Python.

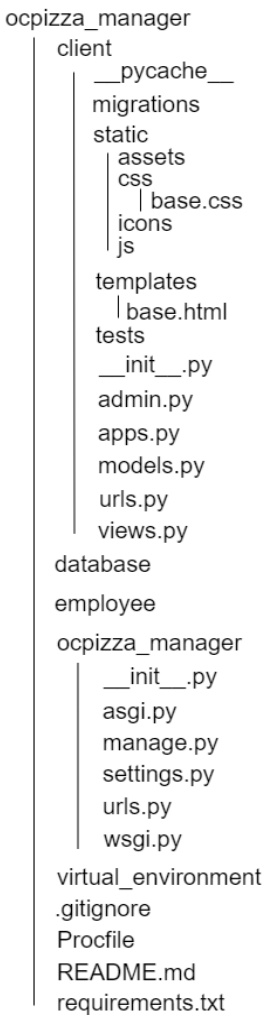
Une couche **Vue** : La vue est une fonction Python servant à écouter les requêtes http sur une route (url) donnée. Elle reçoit donc une requête HTTP et y réponds de manière intelligible par le navigateur. Elle réalise également toutes les actions nécessaires pour répondre à la requête http. Chaque vue est associée à une url et n'a qu'une utilité (au possible).

Une couche **Template** : Le template est un fichier HTML rendu par la vue sous forme de réponse http pouvant recevoir des objets Python. Concrètement le template peut interpréter des variables et les afficher. Par exemple, nous pouvons donner la variable tom="Tom" au template index.html et ce dernier l'affichera à la place du prénom.

### *7.1.2 - Structure des sources*

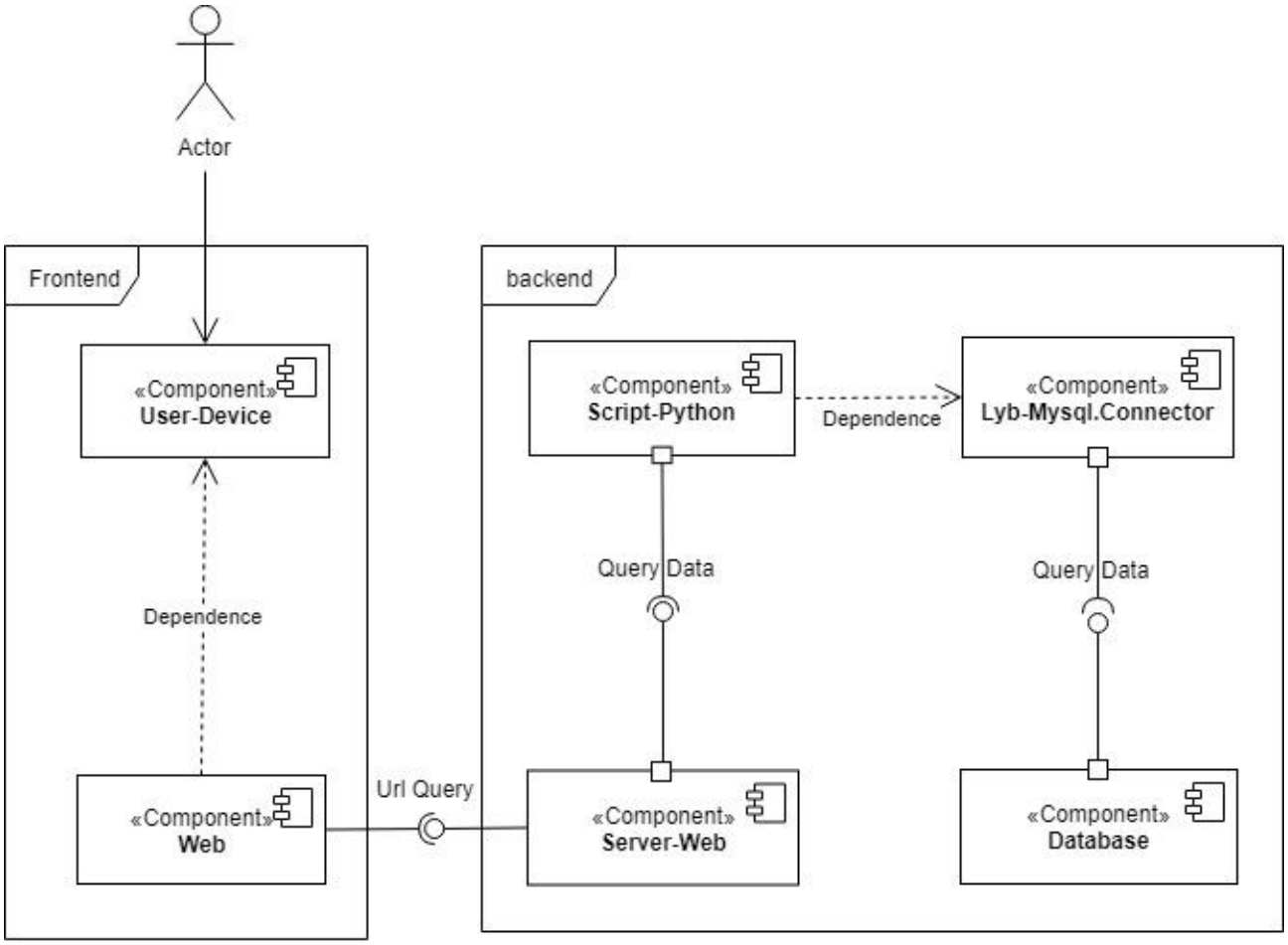
La structuration du répertoire du projet suit la logique suivante :

- Le répertoire source 'ocpizza_manager' contient les applications 'client', 'database' et 'employee'. Sur le même plan nous avons le dossier des settings portant le même nom que le projet 'ocpizza_manager' dans lequel se trouve le 'settings.py' que nous décrirons plus bas ou encore le 'manage.py' chargé d'éxecuter nos commandes Django utile au développement. Ensuite nous avons l'environnement virtuel 'virtual_environment', le '.gitignore' pour ne pas pusher l'environnement virtuel, le 'Procfile' essentiel à la configuration du server sous Heroku, le 'README.md' et pour finir le 'requirements.txt' qui gérera toutes nos dépendances nécessaires au fonctionnement du système.



## 7.2 - Application Web

Voici le diagramme UML de composants montrant les différents modules et leur inter-dépendances.



# 8 - POINTS PARTICULIERS

## 8.1 - Gestion des logs

Les logs sont gérés simplement par Heroku dans l'onglet 'More' qui propose ensuite l'option 'View logs' lorsque l'application est selectionné au préalable. Il est possible de choisir de voir tout les processus tel que le déploiement par exemple en selectionnant 'all processes' ou juste ceux de la partie web en selectionnant 'web'.

## 8.2 - Fichiers de configuration

### *8.2.1 - Application web*

#### *8.2.1.1 - Configurations Heroku*

Tout les paramètres liés à l'application web seront repertoriés dans les settings du compte Heroku afin de pouvoir gérer le système. S'y trouvent en l'occurence toutes les variables d'environnement propre à la production et importantes au fonctionnement d'un projet Django telles que :

OCPIZZA_MANAGER_KEY = 4jdsksnkgndklnd59sknsk6bv6fv

la clé d'authentification de l'application lors d'échanges de données entre le server et un client. Elle est bien évidement différente de celle de l'environnement de développement afin d'éviter toute faille de sécurité.

ENV = PRODUCTION

Cette variable sert à différencier l'environnement de développement de production.

DJANGO_SETTINGS_MODULE = ocpizza_manager.settings

Cette variable informe heroku du path (chemin) qui donne accès aux settings de l'application Django.

DATABASE_URL = postgres://f27djkjfkjgFeD23jdhfk509f3fgh43

Enfin la variable d'accès à la base de données Postgrès, son path.

#### *8.2.1.2 - Datasources*

Heroku met à disposition par le biais de "Add-ons" tout un panel d'options supplémentaires afin de gérer certain organe nécessitant une gestion externe à Heroku comme par exemple dans notre cas la base de données dont nous allons nous servir pour enregistrer toutes les informations liées aux

clients, stock ou encore recettes des pizzas etc. Il est important de noter que nous n'utiliserons qu'une seule base de données pour l'ensemble des restaurants. Il faudra aussi payer ce service car la versions gratuite ne propose que dix-mille lignes en base ce qui dans notre cas ne sera pas suffisant. Nous utiliserons donc le Add-ons "Heroku-Postgrès" qui servira à fournir un accès à une base de données par le biais d'une variable d'environnement que nous avons décris précédemment.

### *8.2.1.3 - Configurations Django*

Ici un aperçu du fichier 'settings.py' de configuration de l'application Django situé dans le repertoire 'ocpizza_manager' du projet 'ocpizza_manager' vu plus haut dans la structure des sources:

```
""" Django settings for ocpizza_manager project.
Generated by 'django-admin startproject' using Django 3.1.3.
For more information on this file, see
https://docs.djangoproject.com/en/3.1/topics/settings/
For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.1/ref/settings/"""

import os
from pathlib import Path
import dj_database_url
import Django_heroku

# Build paths inside the project like this: BASE_DIR / 'subdir'.
# BASE_DIR = Path(__file__).resolve().parent.parent
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# La clé d'authentification de sécurité reste évidemment secrète.
SECRET_KEY = os.environ.get('OCPIZZA_MANAGER_KEY')

# Ici on dit à Django que si la variable d'environnement 'ENV' est égale à
'PRODUCTION' alors on passe en mode DEBUG = True et False si elle est égale à autre
chose. D'où l'importance de bien configurer les variables d'environnement dans
Heroku.
BEBUG = True if os.environ.get('ENV') == 'production' else False

# Ici nous précisons à Django les noms de domaine différents qu'il doit accèpter en
fonction de l'environnement d'éxecution.
```

```
ALLOWED_HOSTS = ['https://ocpizza_manager.herokuapp.com', 'localhost']
# Applications présentes dans le projet, on y retrouve nos trois applications
‘client’, ‘database’ et ‘employee’. Chaque nouvelle application doit être
mentionnée ici, Django ne l’a prendra pas en compte autrement.
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_extensions',
    'client',
    'database',
    'employee',
]
# Le Client qui dialoguera avec le système à ne pas confondre avec le client qui
commandera des pizzas car le client dont on parle ici pourrait très bien être un
employé de la compagnie.
AUTH_USER_MODEL = ['client.ClientUser', 'client.EmployeeUser']

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
# Le chemin qui mène aux urls racines du projet.
ROOT_URLCONF = 'ocpizza_manager.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates',],
        'APP_DIRS': True,
        'OPTIONS': {
                'context_processors': [
                'django.template.context_processors.debug',
```

```
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'users.context_processor.form_renderer'
            ],
        },
    },
]

WSGI_APPLICATION = 'ocpizza_manager.wsgi.application'

# Paramètres de la base de données, ici PostgrèsSQL
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'ocpizza_manager_db',
        'USER': 'postgres',
        'PASSWORD': '##################',
        'HOST': '',
        'PORT': '5432',
    }
}

# Password validation
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityVal
idator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
# Internationalisation
LANGUAGE_CODE = 'fr'
TIME_ZONE = 'UTC'
USE_I18N = True
```

```
USE_L10N = True
USE_TZ = True

# Tous les fichiers static (ex: CSS, JavaScript, Images)
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
# On précise à Django q'une connexion et déconnexion redirigeront toujours sur la
page d'accueil nommée ici 'home'.
LOGIN_REDIRECT_URL = 'home'
LOGOUT_REDIRECT_URL = 'home'

django_heroku.settings(locals())
```

### *8.2.2 - Applications Django*

Un projet Django peut se découper en plusieurs applications lesquelles sont créées dans le but d'isoler une entité unique à part entière centré sur une fonctionnalité qui se démarque des autres. L'avantage second de l'application est sa portabilité possible dans un autre projet. Dans le cas de OC Pizza-Manager on en distinguera donc trois :

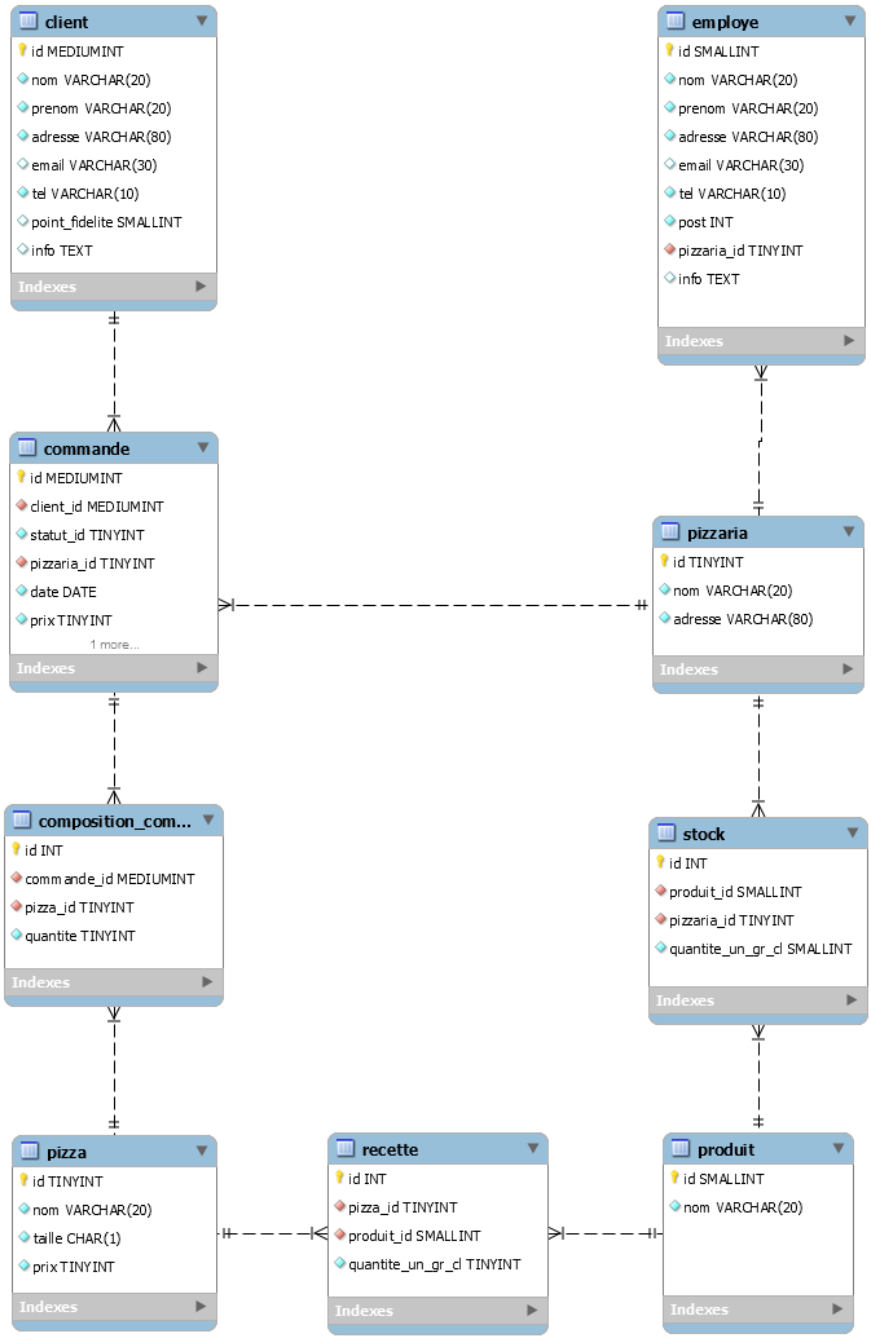
'client' exécutera l'interface à fournir aux clients et mettra en place le modèle 'Client' afin de gérer toutes les informations liées à un client comme son adresse mail ou son mot de passe pour ne citer qu'eux.

'database' qui gérera tous nos modèles en dehors des personnes physiques tels que les pizzas, les commandes, les recettes etc…

'employee' qui fournira les interfaces aux employés et créera le modèle 'employé'. Nous décidons de dissocier les clients des employés par deux applications et modèle différents pour mieux avoir l Chaque employées sera qualifié en fonction de son poste par le champs 'post' qui portera l'une des valeurs suivantes : 0 pour le standardiste, 1 pour le pizzaiolo, 2 le livreur et enfin 3 le gérant, ceci dans le but d'associer de manière propre chaque interface pour chaque poste.

## 8.3 - Ressources

Voici donc le modèle physique de données dont nous disposons:



## 8.4 - Environnement de développement

L'intégralité du programme sera hebergé sur Github de manière privée donc non accessible au public afin que le client en garde l'exclusivité et le contrôle. La société Openclassrooms donnera accès au programme source aux techniciens chargé de sa maintenance et amelioration sous l'accord contractuel préalable de la société OC Pizza qui en possèdera les clés d'accès. L'environnement de déploiement sur le server Heroku sera lui aussi géré de la même manière et sous les mêmes conditions.

## 8.5 - Les Tests

Chaque module et fonctionnalitée sera testé automatiquement afin de s'assurer que tout fonctionne et qu'aucune erreur n'intervient dans l'intégration d'une nouvelle fonctionnalité. Nous utiliserons pour cela Pytest pour automatiser le lancement de tout les tests et de la librairie 'unittest' mise à disposition par Django pour les tests unitaires. Nous utiliserons également Sélenium pour automatiser les tests fonctionnels afin de gagner du temps.

## 8.6 - Procédure de packaging / livraison

Comme expliqué précédement l'application "OC Pizza-Manager" sera sous forme de projet privé sur le cloud Github en version finale et déployé ensuite sur Heroku sous cette même version. Version qui avant d'être mis dans le stage, commitée et pushée sur Github suivra une batterie de tests afin de s'assurer du fonctionnement. Le déploiement sur Heroku pour la mise en production ne se fera donc qu'après avoir suivit ce protocole : test, sauvegarde, déploiement.