OC Pizza

Système de gestion de pizzeria en ligne

Dossier de conception technique

Version 1.0

Auteur
Clémence Robin
Analyste-programmeur

IT Consulting & Development
7 cité Paradis – 01 02 03 04 05 – contact@it-solutions.fr
www.it-solutions.fr
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
Clémence R.	01/09/2020	Création du document	1.0

2 - Introduction

2.1 - Objet du document

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

Ce document est destiné aux développeurs, mainteneurs et à l'équipe technique de OC Pizza.

L'objectif du document est de présenter l'architecture technique mises en œuvre afin de réaliser l'application.

Les éléments du présents dossiers découlent :

- de nos entretiens avec le client pour répondre à ses attentes ainsi que
- du document de spécifications fonctionnelles qui en est résulté

2.2 - Références

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

1. **DCF – 1.0** : Dossier de conception fonctionnelle de l'application
2. **DE – 1.0** : Dossier d'exploitation de l'application

3 - Architecture de composants

3.1 - Composants de l'application

Le site web est représenté par un sous-système contenant lui même trois composants : *Internal search engine*, *Cart* et *Authentication*. Le composant *Internal search engine* expose l'interface *Product search*, permettant ainsi de rechercher les produits en utilisant l'interface *Search in stocks* fournie par le composant *Stock* du sous-système *Restaurant Stock*.

Le composant *Cart* utilise l'interface *Manage Orders* founie par le composant *Orders* lors du paiement et fournit l'interface *Online Ordering*.

Le composant *Authentication* permet quant à lui aux utilisateurs de créer un compte, s'authentifier ou se déconnecter en fournissant l'interface *User sessions*.

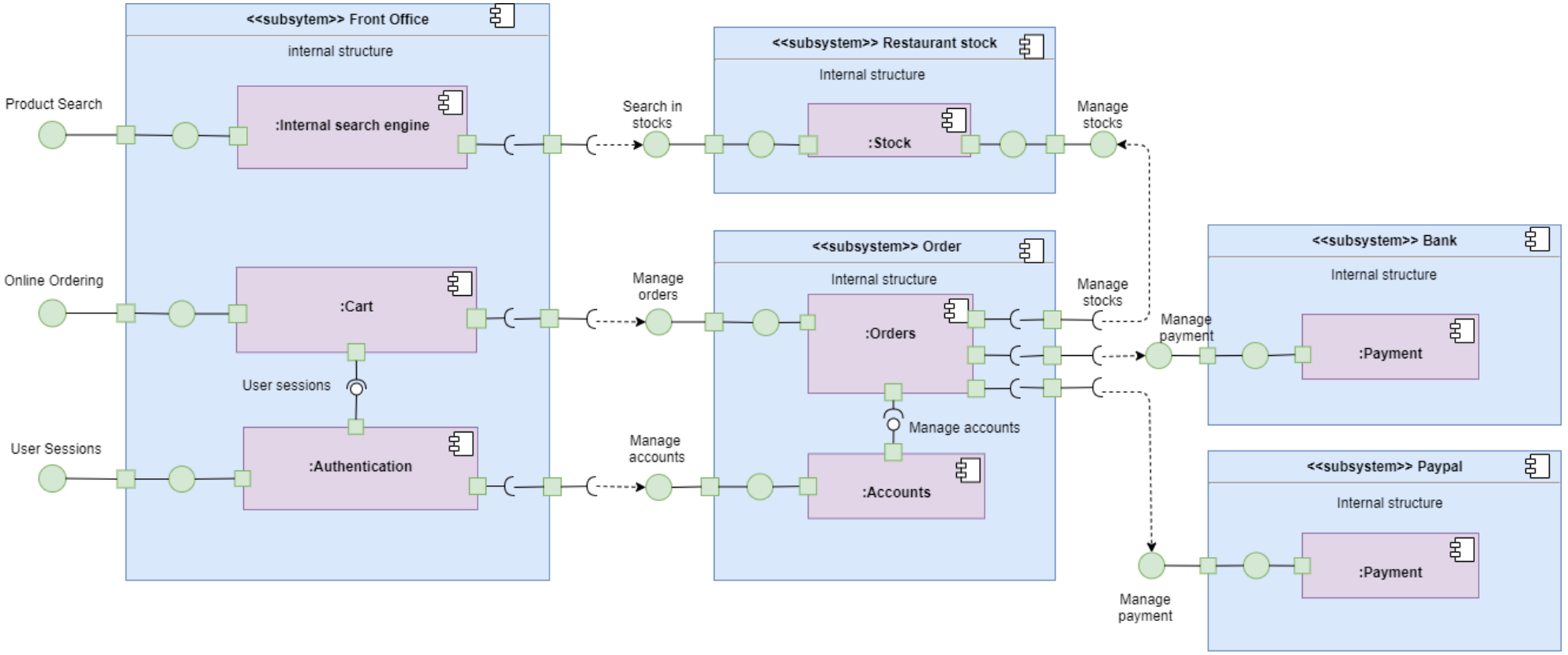
Le sous-système *Order* fournit deux interfaces, *Manage accounts* et *Manage orders*. Les connecteurs de délégation lient les objets externes à ce sous-système aux composants *Orders* et *Accounts* appartenant à ce système.

Le sous-système *Restaurant Stock* fournit deux interfaces *Search in stocks* et *Manage Stocks*.

Les deux sous-système *Bank* et *Paypal* fournissent chacun une interface *Manage Payment* : l'interface *Manage Orders* exposée par le composant *Accounts* peut ainsi joindre les interfaces *Manage Payment*.

Le parcours d'une commande pourrait donc être le suivant :

- L'utilisateur est sur le site web (sous-système Front office).
- Il recherche des produits : le moteur de recherche fait appel aux stocks pour vérifier la disponibilité de ces derniers (sous-système Restaurant Stock).
- L'utilisateur rajoute des produits dans le panier (sous-système Front-Office).
- Il passe commande : il n'est pas authentifié donc la commande ne peut pas être liée à un utilisateur (sous-système Orders).
- Il s'authentifie (sous-système Front-Office).
- Il tente à nouveau de passer commande (sous-système Orders).
- Le système de commande lui demande ses données bancaires ou les informations de son compte Paypal (sous-systèmes Paypal et Bank).
- En cas de succès, les produits commandés sont retirés de l'inventaire du stock (sous-système Restaurant Stock).


<<subsytem>> Front Office
internal structure
Product Search
:Internal search engine
Online Ordering
:Cart
User sessions
User Sessions
:Authentication
Search in stocks
<<subsystem>> Restaurant stock
Internal structure
:Stock
Manage stocks
Manage orders
<<subsystem>> Order
Internal structure
:Orders
Manage accounts
Manage accounts
:Accounts
Manage stocks
Manage payment
<<subsystem>> Bank
Internal structure
:Payment
<<subsystem>> Paypal
Internal structure
:Payment
Manage payment

4 - Architecture de Déploiement

4.1 - Déploiement de l'application

4.1.1 - Matériel client

L'utilisateur pourra communiquer avec l'application du restaurant par le biais de différents appareils devant être connectés à internet et posséder un navigateur web pour que ce dernier puisse faire les requêtes au serveur contenant l'application. Les requêtes se feront par le biais du langage de protocole HTTPS et de l'API de l'application du restaurant. Le navigateur affichera le HTML et le CSS et exécutera les fonctions de JavaScript et de JQuery.

4.1.2 - Serveur d'application

Le serveur d'application sera un serveur privé virtuel (VPS) fonctionnant avec Debian 10. Il hébergera :

- Le serveur web HTTP Nginx (low-client) pour traiter les contenus statiques et agir en tant que proxy inverse. Il reçoit les requêtes HTTP et les transfère au serveur d'application Gunicorn. Il fait aussi parvenir les réponses HTTP de l'application Django au navigateur web du client.
- Le serveur d'application HTTP Gunicorn (fast-client) qui travaille en association avec Nginx. Lorsque Nginx reçoit une requête dynamique, il transfère la requête à Gunicorn via un socket.
- L'application web du restaurant, fonctionnant avec le framework Django 3. Django reçoit les requêtes de Nginx et lui renverra les réponses HTTP.
- Le système de cache Redis, qui servira à réduire le temps de chargement de l'application lorsque la machine aura déjà les informations en mémoire cache.

Le serveur d'application communiquera avec le serveur de base de données avec le langage de procédure chargeable PL/pgSQL pour le système de base de données PostgreSQL.

Le serveur hébergeant l'application pourra communiquer, outre le serveur hébergeant la base de données, avec deux autres serveurs afin d'effectuer les transactions bancaires :

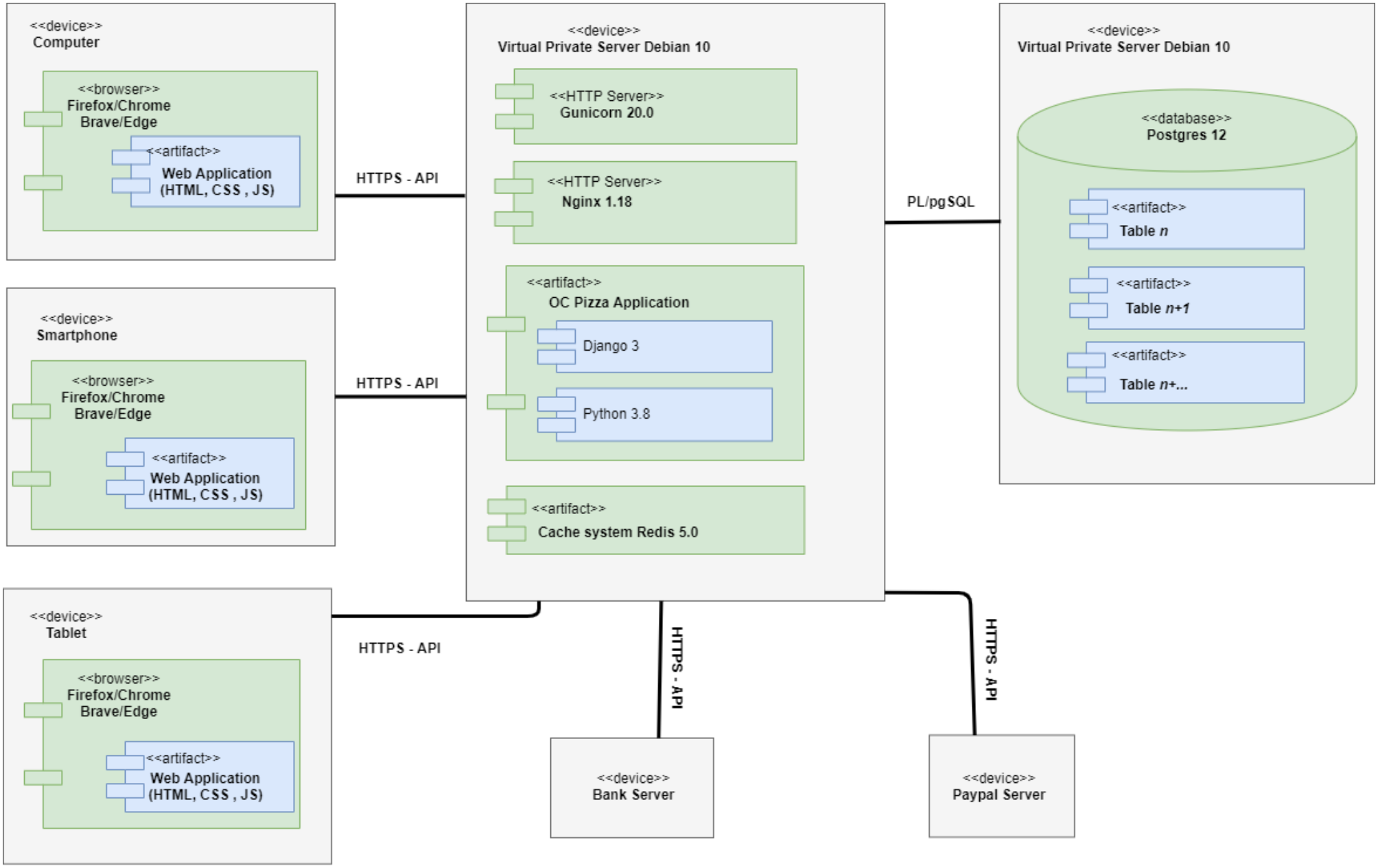
- Le serveur de la banque cible qui effectuera les procédures de vérifications et de transactions, par le le biais du langage de protocole HTTPS et de l'API fournie par la banque.
- Le serveur de Paypal qui effectuera les procédures de vérifications et de transactions, par le le biais du langage de protocole HTTPS et de l'API fournie par Paypal.

4.1.3 - Serveur de base de données

Le serveur de base de données sera un serveur virtuel privé (VPS). Il hébergera la base de données PostgreSQL 12.

4.1.4 - Diagramme de déploiement

Le diagramme de déploiement ci-dessous décrit les différents éléments prenant part à la production de l'application.


<<device>>
Computer
<<browser>>
Firefox/Chrome
Brave/Edge
<<artifact>>
Web Application
(HTML, CSS , JS)
HTTPS - API
<<device>>
Virtual Private Server Debian 10
<<HTTP Server>>
Gunicorn 20.0
<<HTTP Server>>
Nginx 1.18
<<artifact>>
OC Pizza Application
Django 3
Python 3.8
<<artifact>>
Cache system Redis 5.0
PL/pgSQL
<<device>>
Virtual Private Server Debian 10
<<database>>
Postgres 12
<<artifact>>
Table n
<<artifact>>
Table n+1
<<artifact>>
Table n+...
<<device>>
Smartphone
<<browser>>
Firefox/Chrome
Brave/Edge
<<artifact>>
Web Application
(HTML, CSS , JS)
HTTPS - API
<<device>>
Tablet
<<browser>>
Firefox/Chrome
Brave/Edge
<<artifact>>
Web Application
(HTML, CSS , JS)
HTTPS - API
HTTPS - API
HTTPS - API
<<device>>
Bank Server
<<device>>
Paypal Server

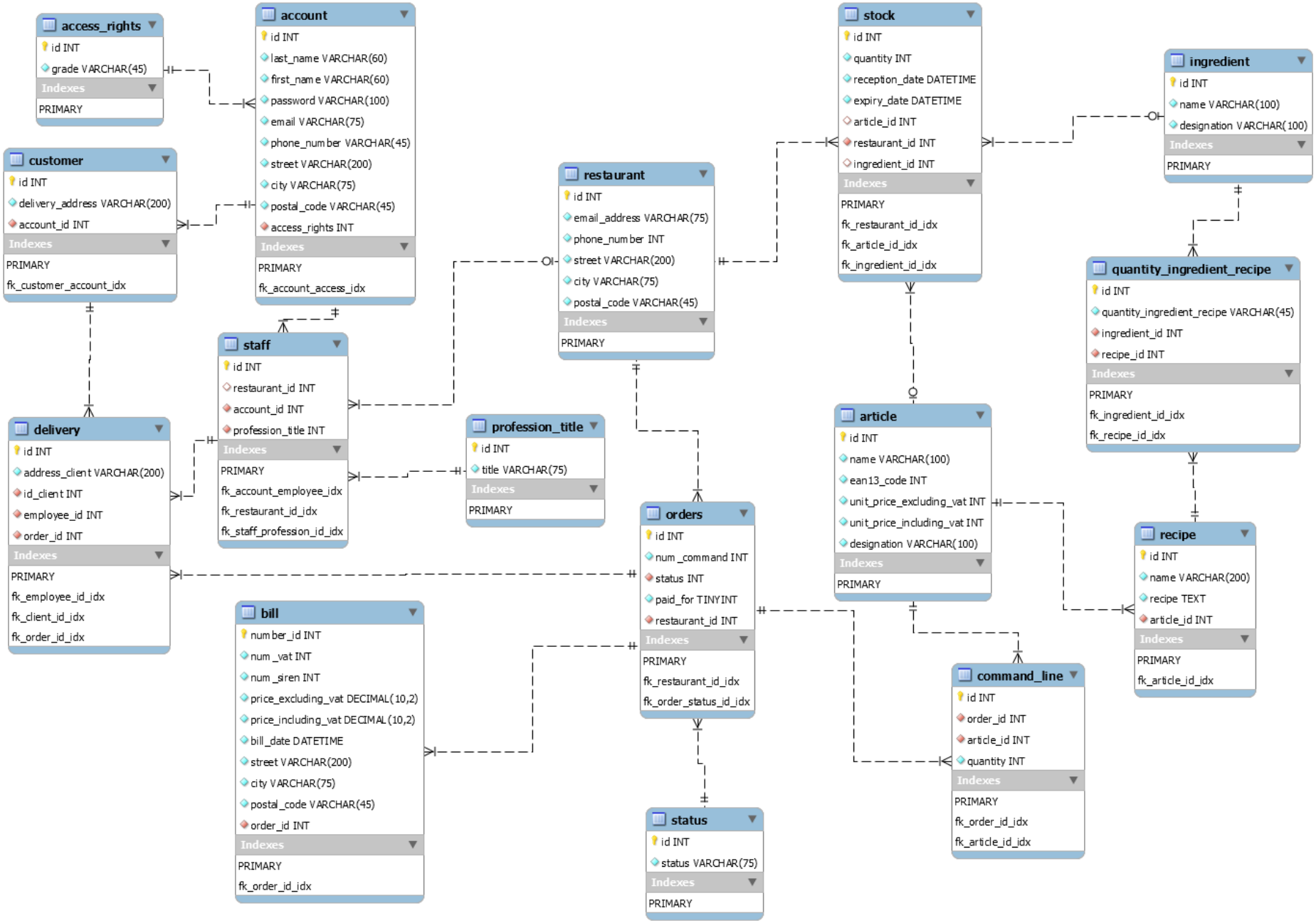
4.2 - Serveur de Base de données

Le serveur de base de données sera aussi un serveur privé virtuel. Il hébergera :

- Le SGBDR PostgreSQL 12
- La plateforme d'administration de base de données Pgadmin 4, qui facilitera la gestion de la base de donnée PostgreSQL.

4.2.1 - Modèle physique de données

Nous pouvons voir ci-dessous les colonnes de chaque table, leur type et les liens entre les tables.


access_rights
id INT
grade VARCHAR(45)
Indexes
PRIMARY
account
id INT
last_name VARCHAR(60)
first_name VARCHAR(60)
password VARCHAR(100)
email VARCHAR(75)
phone_number VARCHAR(45)
street VARCHAR(200)
city VARCHAR(75)
postal_code VARCHAR(45)
access_rights INT
Indexes
PRIMARY
fk_account_access_idx
stock
id INT
quantity INT
reception_date DATETIME
expiry_date DATETIME
article_id INT
restaurant_id INT
ingredient_id INT
Indexes
PRIMARY
fk_restaurant_id_idx
fk_article_id_idx
fk_ingredient_id_idx
ingredient
id INT
name VARCHAR(100)
designation VARCHAR(100)
Indexes
PRIMARY
customer
id INT
delivery_address VARCHAR(200)
account_id INT
Indexes
PRIMARY
fk_customer_account_idx
restaurant
id INT
email_address VARCHAR(75)
phone_number INT
street VARCHAR(200)
city VARCHAR(75)
postal_code VARCHAR(45)
Indexes
PRIMARY
quantity_ingredient_recipe
id INT
quantity_ingredient_recipe VARCHAR(45)
ingredient_id INT
recipe_id INT
Indexes
PRIMARY
fk_ingredient_id_idx
fk_recipe_id_idx
staff
id INT
restaurant_id INT
account_id INT
profession_title INT
Indexes
PRIMARY
fk_account_employee_idx
fk_restaurant_id_idx
fk_staff_profession_id_idx
profession_title
id INT
title VARCHAR(75)
Indexes
PRIMARY
delivery
id INT
address_client VARCHAR(200)
id_client INT
employee_id INT
order_id INT
Indexes
PRIMARY
fk_employee_id_idx
fk_client_id_idx
fk_order_id_idx
article
id INT
name VARCHAR(100)
ean13_code INT
unit_price_excluding_vat INT
unit_price_including_vat INT
designation VARCHAR(100)
Indexes
PRIMARY
orders
id INT
num_command INT
status INT
paid_for TINYINT
restaurant_id INT
Indexes
PRIMARY
fk_restaurant_id_idx
fk_order_status_id_idx
recipe
id INT
name VARCHAR(200)
recipe TEXT
article_id INT
Indexes
PRIMARY
fk_article_id_idx
bill
number_id INT
num_vat INT
num_siren INT
price_excluding_vat DECIMAL(10,2)
price_including_vat DECIMAL(10,2)
bill_date DATETIME
street VARCHAR(200)
city VARCHAR(75)
postal_code VARCHAR(45)
order_id INT
Indexes
PRIMARY
fk_order_id_idx
command_line
id INT
order_id INT
article_id INT
quantity INT
Indexes
PRIMARY
fk_order_id_idx
fk_article_id_idx
status
id INT
status VARCHAR(75)
Indexes
PRIMARY

4.2.1.1 - Descriptions du modèle physique de données

Nous présentons ici sur les tables ainsi les clés étrangères qui les lient entre elles. Lorsqu'une clé est abordée dans la description d'une table A envers une table B, nous ne reviendrons par sur cette relations lors de la description de la table B.

4.2.1.1.1 stock

Représente les stocks d'un restaurant.

Les champs articles_id et ingredient_id peuvent tous les deux être *null*. En effet, une instance du stock peut concerner un ingredient (matière première) ou un article (ex :cannette de soda). L'un des deux champs sera donc *null*.

La table contient trois clés étrangères :

- **fk_stock_restaurant_id** : entre le champ restaurant_id de la table *stock* et le champ id de la table *restaurant*
- **fk_stock_article_id** : entre le champ article_id de la table *stock* et le champ id de la table *article*
- **fk_stock_ingredient_id** : entre le champs ingredient_id de la table *stock* et le champ id de la table *ingredient*

4.2.1.1.2 recipe

Représente les recettes des articles confectionnés par l'entreprise.

Une recette doit correspondre à un article préparé par le restaurant, donc à un instance de la classe *article*. Elle aura donc une clé étrangère entre le champ article_id et le champ id de la table *article* :

- **fk_recipe_article_id** : entre le champ article_id de la table *recipe* et le champ id de la table *article*

4.2.1.1.3 orders

Représente les commandes/

Chaque commande doit être associée à un restaurant et chaque commande doit être associée à un statut. La table *orders* contient donc deux clés étrangères :

- **fk_order_restaurant_id** : entre le champ restaurant_id de la table *orders* et le champs id de la table *restaurant*
- **fk_order_status_id** : entre le champ status de la table *orders* et le champs id de la table *status*

4.2.1.1.4 command_line

La table *command_line* sert à préciser le nombre d'article commandé pour une commande donnée. Elle doit donc être associée à l'id de l'article et l'id de la commande. La table contient par conséquent deux clés étrangères:

- **fk_command_order_id** *:* entre le champ order_id de la table *command_line* et le champ id

de la table *orders*

- **fk_command_article_id** *:* entre le champ article_id de la table *command_line* et le champ id de la table *article*

4.2.1.1.5 bill

Représente les factures des commandes.

Chaque facture devant être associée à une commande, la table Bill contient donc une clé étrangère :

- **fk_bill_order_id** : entre le champ order_id de la table *bill* et le champ id de la table *orders*

4.2.1.1.6 account

Représente les comptes des utilisateurs.

Les comptes pouvant être associés à des clients ou bien à des employés ayant des droits différents, chacun d'entre eux doit être associé à un droit spécifique. Nous avons donc une clé étrangère dans la table *account* :

- **fk_account_access** : entre le champ access_rights de la table *account* et le champ id de la table *access_rights*

4.2.1.1.7 quantity_ingredient_recipe

Représente les quantités à utiliser pour un ingrédient donné.

Cette table servant à indiquer les quantités d'un ingrédient donné pour une recette donnée, nous avons choisi de donner au champ *quantity_ingredient_recipe* le type VARCHAR : cela permet d'insérer d'autres types de données que des chiffres, par exemple : « Un zeste de » ou « Une cuillère à soupe de ... ». La table *quantiy_ingredient_recipe* faisant le lien entre un ingrédient et une recette, elle contient par conséquent deux clés étrangères :

- **fk_quantity_ingredient_id** : entre le champ ingredient_id de la table *quantity_ingredient_recipe* et le champ id de la table *ingredient*
- **fk_quantiry_recipe_id** : entre le champ recipe_id de la classe *quantity_ingredient_recipe* et le champ id de la table *recipe*

4.2.1.1.8 customer

Représente les clients.

Chaque client passant commande doit avoir un compte associé. Nous avons donc une clé étrangère dans la table Customer :

- **fk_customer_account_id** : entre le champ account_id de la table *customer* et le champ id de la table *account*

4.2.1.1.9 staff

Représente les membres du personnel.

Nous avons choisi de laisser la possibilité au champ restaurant_id d'avoir la valeur NULL, selon si un employé, comme un livreur, travaille dans un ou plusieurs restaurants. De plus, le directeur n'est pas

associé à un restaurant en particulier.

Chaque membre du personnel possède un compte et est associé à un intitulé de profession. De même, si le champ restaurant_id n'est pas *null*, il doit être associé à un restaurant donné. Nous avons donc trois clés étrangères :

- **fk_staff_account_id** : entre le champ account_id de la table *staff* et le champ id de la table *account*
- **fk_staff_restaurant_id** : entre le champ restaurant_id de la table *staff* et le champ id de la table *restaurant*
- **fk_staff_profession_id** : entre le champ profession_id de la table *staff* et le champ id de la table *profession_title*

4.2.1.1.10 delivery

Représente les livraisons des commandes.

Chaque livraison correspondant à une commande étant livrée à un client par un livreur donné, nous avons trois clés étrangères dans la table delivery :

- **fk_delivery_employee_id** : entre le champ employee_id de la table *delivery* et le champ id de la table *staff*
- **fk_delivery_client_id** : entre le champ id_client de la table *delivery* et le champ id de la table *customer*
- **fk_delivery_order_id** : entre le champ *order_id* de la table *delivery* et le champ id de la table *orders*

4.2.1.1.11 restaurant

Représente les restaurants de l'entreprise.

4.2.1.1.12 profession_title

Représente l'intitulé de poste d'un employé.

4.2.1.1.13 access_rights

Représente les droits d'accès d'un compte selon s'il appartient à un client ou à un membre du personne, et, dans ce dernier cas, le niveau d'accès de l'employé.

Note : Plusieurs choix s'offrent pour représenter les droits d'accès d'utilisateur. Il est possible de définir les listes de contrôle d'accès (ACL) dans la base de données PostgreSQL. Par soucis de clarté et de simplicité, l'application vérifiera les droits d'accès de l'utilisateur connecté en effectuant une requête à la base de donnée. Il est aussi possible de définir des groupes et leurs permissions associées dans l'interface administrateurs de Django. Les tables associées (auth_group, auth_user, auth_permission …) sont créées par Django.

4.2.1.1.14 ingredient

Représente les ingrédients utilisés dans les articles confectionnés par l'entreprise.

4.2.1.1.15 status

Représente les différents statuts d'avancement d'une commande.

4.2.1.1.16 article

Représente les articles vendus par l'entreprise.

4.3 - Serveurs de paiement

Lors du paiement, le serveur d'application communiquera avec des serveurs externes à notre infrastructure :

- les serveurs bancaires
- les serveurs de Paypal

5 - Architecture logicielle

5.1 - Principes généraux

Les sources et versions du projet sont gérées par Git.

Le développement de l'application se présente sous la forme d'un projet Django, lui-même divisé en plusieurs applications.

5.1.1 - Les couches

L'architecture applicative est la suivante :

- une couche **présentation** : interface client de l'application se trouvant dans le fichier *views.py*
- une couche **métier** : responsable de la logique métier du composant se trouvant dans le fichier *models.py*
- une couche **accès base de données** : les opérations de lecture écriture sur la base de données sont faites par l'ORM de Django qui communique avec la base de donnée

5.1.2 - Les modules

L'application est divisée en 4 modules gérant chacune une couche de l'application.

- L'authentification
- Le stock
- L'administration
- Les commandes

5.1.3 - Structure des sources

La structure est créée de façon à respecter les conventions de Django. La structuration des répertoires du projet suit la logique suivante :

```
oc_pizza/
├── administration
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── static
│   ├── templates
│   ├── tests.py
│   ├── urls.py
│   └── views.py
```

```
├── auth
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── static
│   ├── templates
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── manage.py
├── oc_pizza
│   ├── asgi.py
│   ├── __init__.py
│   ├── settings
│   │   ├── __init__.py
│   │   ├── local.py
│   │   └── production.py
│   ├── urls.py
│   └── wsgi.py
├── order
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── static
│   ├── templates
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── README.md
├── requirements.txt
├── staticfiles
├── stock
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── static
│   ├── templates
│   ├── tests.py
│   ├── urls.py
│   └── views.py
└── tests
```

6 - Points particuliers

6.1 - Gestion des logs

Le projet Django Sentry est une application de monitoring qui permet de gérer les logs et d'être averti des dysfonctionnements.

6.1.1 - Création de compte chez Sentry

OC Pizza doit s'inscrire sur le site de Sentry.

6.1.2 - Configuration de Sentry

Lors de l'enregistrement du compte, Sentry affichera les informations à rajouter dans *oc_pizza/oc_pizza/settings/__init__.py*.

Un tableau de bord permettra ensuite d'afficher et de gérer plusieurs informations, telles que les logs, la performance ou encore la création d'alertes.

6.2 - Fichiers de configuration

6.2.1 - Application web

La configuration de l'application se trouve dans le dossier *settings*, composé de trois fichiers :

- __init__.py : rassemble toutes les configurations commune à l'environnement de production et à l'environnement de développement
- local.py : rassemble toutes les configurations propres au développement, comme l'activation du mode de débogage.
- production.py : rassemble toutes les configurations propres à l'environnement de production.

6.3 - Ressources

6.3.1 - Charte graphique

Le cahier des charges graphique nous est fourni par OC Pizza.

6.3.2 - Données

Les données nécessaires au fonctionnement de la base de données (comptes, recettes, etc) nous sont fournies par OC Pizza.

6.4 - Environnement de développement

Le développement de l'application ne nécessite pas l'utilisation d'un IDE en particulier. Cependant, l'édition professionnelle de PyCharm rencontre tous les avantages liés à un développement couplant Django et Python.
Un environnement virtuel sera créé pour s'assurer de la compatibilité entre tous les composants de l'application.

6.5 - Procédure de packaging / livraison

6.5.1 - Serveur d'application web

L'application fera l'objet d'un déploiement sur un serveur privé virtuel loué chez Digital Ocean.
Digital Ocean permet d'obtenir une machine virtuelle où l'on peut installer Debian. De plus, ressources s'adaptent au besoin de l'application.

Le produit qui correspond à nos besoin se trouve dans le catalogue "General Purpose Droplet", qui permet d'avoir un CPU dédié et non partagé. Il est conçu pour les applications web avec un certain niveau de trafic, les sites de commerce en ligne, ainsi que les bases de données de taille moyenne.

Si les besoins d'OC Pizza évolue, la machine adaptera ses ressources et le prix s'adaptera en conséquence.
Notre machine partira donc avec ces caractéristiques :

- 8GB de RAM
- 4vCPU
- 5 TB de transfert de données
- 160 GB d'espace disque
- Coût : 40$

6.5.2 - Serveur de base de données

Comme l'application web, la base de donnée sera aussi déployée sur un serveur loué chez Digital Ocean. Digital met à disposition des machines virtuelles avec bases de données gérées, dont PostgreSQL. Elles permettent notamment de s'affranchir de l'installation, de la maintenance ainsi que d'automatiser les sauvegardes et les mises à jours.
La base de donnée est joignable avec l'API.

Notre machine aura ces caractéristiques :

- 1GB de RAM
- 1vCPU
- 10 GB d'espace disque

- Coût : 15$ par mois.

Il est possible d’allouer plus de ressources si l’évolution de la base de données le nécessite.

6.5.3 - Dossier d’exploitation

OC Pizza se verra remettre un dossier d’exploitation permettant la continuité de l’utilisation de l’application.

7 - Glossaire

SGBDR	Système de gestion de base de données relationnelles
vCPU	CPU virtuel alloué à une machine virtuelle
RAM	Random Access Memory – Mémoire à accès aléatoire
API	Application Programming Interface – Interface de programmation d'application
ACL	liste de contrôle d'accès (Access Control List)
VPS	serveur privé virtuel (Virtual Private Server)