OC Pizza

Système de gestion informatique des pizzerias du groupe OC Pizza

Dossier de conception technique

Version 1.0

Auteur
David Bouzerar
Analyste-Programmeur

IT Consulting & Development

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
David Bouzerar	26/03/2021	Création du document	0.1
David Bouzerar	28/03/2021	Ajout de plusieurs rubriques	0.5
David Bouzerar	07/04/2021	Finalisation 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, basé sur les informations présentent dans le dossier de conception fonctionnelle.

L'objectif de ce document est de présenter la solution technique que nous proposons pour répondre aux besoins exprimés par le client.

2.2 - Références

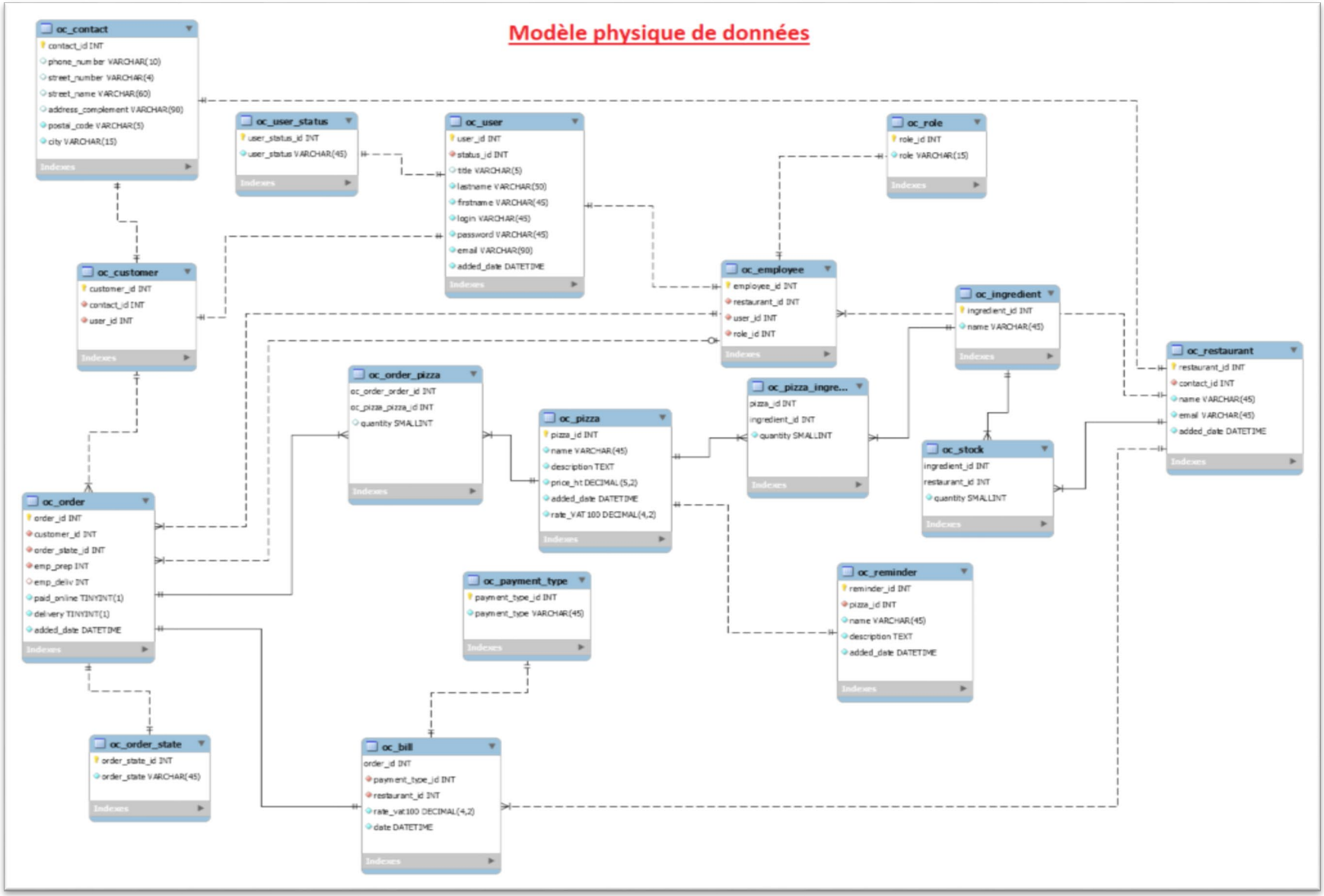
Pour de plus amples informations, se référer également aux éléments suivants :

1. **DCF-01** : Dossier de conception fonctionnelle de l'application
2. **DEXP-03** : Dossier d'exploitation de l'application
3. **PV-04** : PV de livraison

3 - ARCHITECTURE TECHNIQUE

3.1 - Modèle physique de données

Modèle physique de données



3.2 - Description du modèle physique de données

Le diagramme ci-dessus (*MPD*) représente l'architecture de la base de données et également les relations qu'il existe entre les différentes tables.

De la même manière que le *diagramme de classe*, nous pouvons ici le diviser en trois parties distinctes, dont voici les tables principales :

3.2.1 - Gestion des utilisateurs :

- OC_User : La table « **OC_User** » est utilisée pour stocker les informations d'identification des différents utilisateurs présents dans la base de données. Cette table contient notamment les clients, mais également les employés du groupe. Afin de différencier ces deux types d'utilisateur, il existe un champ « **status_id** » lié à la table « **OC_User_Status** ». Ce champ représente les différents types d'utilisateur énumérés dans la table « **OC_User_Status** ».

- OC_Customer : Les champs de la table « **OC_Customer** » sont liés aux utilisateurs « **clients** » de la table « **OC_User** ». C'est-à-dire, ceux qui ont un « **status_id** » équivalent à « **Customer** » (voir les explications de la table « **OC_User** »). La table « **OC_Customer** » est également liée à la table « **OC_Contact** » qui contient les coordonnées des clients.

- OC_Employee : De la même façon que la table « **OC_Customer** », la table « **OC_Employee** » est liée aux utilisateurs « **employés** » de la table « **OC_User** », et donc, ceux qui ont un « **status_id** » équivalent à la valeur « **OC_Employee** ». Un champ « **role_id** » est également présent dans cette table, et lié à la table « **OC_Role** » qui énumère les différentes fonctions au sein du groupe. Un champ « **restaurant_id** » est également présent pour identifier l'établissement d'accueil de l'employé.

3.2.2 - Gestion des commandes :

- OC_Order : La table « **OC_Order** » contient les informations des commandes passées par les clients. Elles sont identifiées et liées à un client par le champ « **customer_id** ». Un autre champ « **order_state_id** » représente l'état actuel de la commande. Il est lié à la table « **OC_Order_State** » qui énumère les différents états d'une commande (en cours de préparation, livrée etc..).

  Deux liaisons à la table « **OC_Employee** » sont également visibles. Un premier champ nommé « **emp_prep** » et lié à « **employee_id** » permet d'identifier l'employé chargé de préparer la commande. Ce champ ne peut pas être « **null** » dans la mesure ou un préparateur est obligatoire pour chaque commande. Contrairement au livreur identifié ici par le champ nommé « **emp_deliv** » et lié à « **employee_id** », qui lui peut avoir une valeur « **null** ». La livraison n'étant pas obligatoire lors du passage d'une commande.

- OC_Bill : Cette table contient les informations de facturation des différentes commandes. Un champ « **order_number** » contient le numéro de commande lié à la facture. Le champ « **payment_id** » lié à la table « **OC_Payment_Type** » permet d'identifier le type de paiement utilisé (CB, espèce, chèque) énuméré dans cette même table. Le champ « **restaurant_id** » permet d'identifier l'établissement d'origine de la facture.

3.2.3 - Gestion des produits :

- OC_Pizza : Cette table contient les différents produits proposés par le groupe. Outre le nom et la description du produit, on y retrouve également son prix hors-taxe à l'unité et la *TVA* à appliquer. Elle est également liée à la table « **OC_Order** » par l'intermédiaire d'une relation *many-to-many* avec la table intermédiaire nommée « **OC_Order_Pizza** ». Cette dernière permettant de lier un ou plusieurs produits à une commande, et en quantité définie par le champ « **quantity** » dans cette même table.

- OC_Ingredient : La table « **OC_Ingredient** » contient tous les ingrédients utilisés pour confectionner les différentes pizzas proposées. On y trouve un « **id** » et le nom de l'ingrédient. Elle est liée à la table « **Pizza** » par l'intermédiaire d'une *relation many-to-many* avec autre table nommée « **OC_Pizza_Ingredient** », permettant notamment d'identifier les ingrédients utilisés pour chaque pizza et en quantité définie.

- OC_Stock : Cette table liée à « **OC_Ingredient** » permet de connaitre la disponibilité des différents ingrédients ainsi que la quantité restante. De manière à identifier les pizzas encore réalisables à n'importe quel moment du service. En plus de « **ingredient_id** », la clé primaire composée contient également « **restaurant_id** » qui permet d'associer un ingrédient à un point de vente en particulier. Nous pourrons donc avoir, par exemple :

- Dans cet exemple, un même ingrédient apparait dans trois points de vente différents.

Table Stock		
Ingredient_id	**Restaurant_id**	**quantité**
1	1	187
1	2	329
1	3	67

- OC_Restaurant : Cette table permet d'identifier les différents points de vente. Elle est également liée à la table « **OC_Contact** » par l'intermédiaire du champ « **contact_id** » pour retrouver les coordonnées des différents points de vente.

- OC_Reminder : « **OC_Reminder** » est la table utilisée pour l'aide-mémoire, contenant les recettes des produits proposés. Elle est destinée à être utilisée (si nécessaire) par les pizzaïolos lors de la réalisation des pizzas.

3.3 - Diagramme de composants

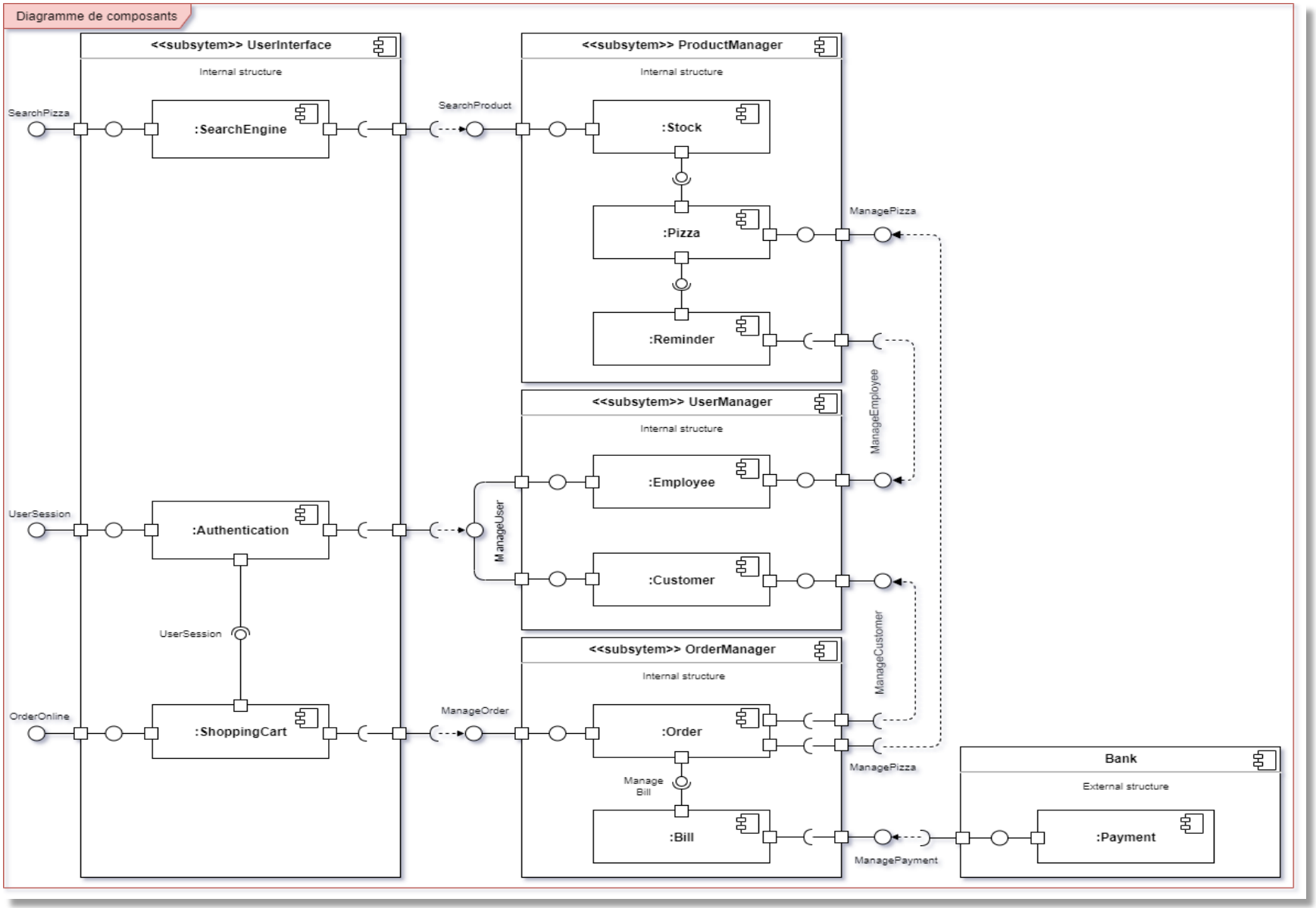


Diagramme de composants
<<subsytem>> UserInterface
Internal structure
SearchPizza
:SearchEngine
SearchProduct
UserSession
:Authentication
UserSession
OrderOnline
:ShoppingCart
ManageOrder
<<subsytem>> ProductManager
Internal structure
:Stock
:Pizza
:Reminder
ManagePizza
<<subsytem>> UserManager
Internal structure
ManageUser
:Employee
:Customer
ManageEmployee
ManageCustomer
<<subsytem>> OrderManager
Internal structure
:Order
Manage Bill
:Bill
ManagePizza
ManagePayment
Bank
External structure
:Payment

3.4 - Description du diagramme de composants

3.4.1 - SearchEngine

Dans cette partie, nous voyons que lors de la recherche d'une pizza, par un client par exemple, le système fait appel au « **ProductManager** » et notamment au composant « **:Stock** » lui-même lié au composant « **:Pizza** ».

3.4.2 - Authentication

Cette partie illustre l'authentification d'un utilisateur. On peut y voir notamment le moment où l'utilisateur est défini comme étant un employé ou un client « **ManageUser** ».

- Employee : Nous voyons ici la relation entre « **:Employee** » et « **:Reminder** » qui lui se trouve dans « **ProductManager** »

- Customer : Nous mettons, ici, en évidence la relation qu'il existe entre « **:Customer** » et « **:Order** » (situé dans « **OrderManager** »).

3.4.3 - ShoppingCart

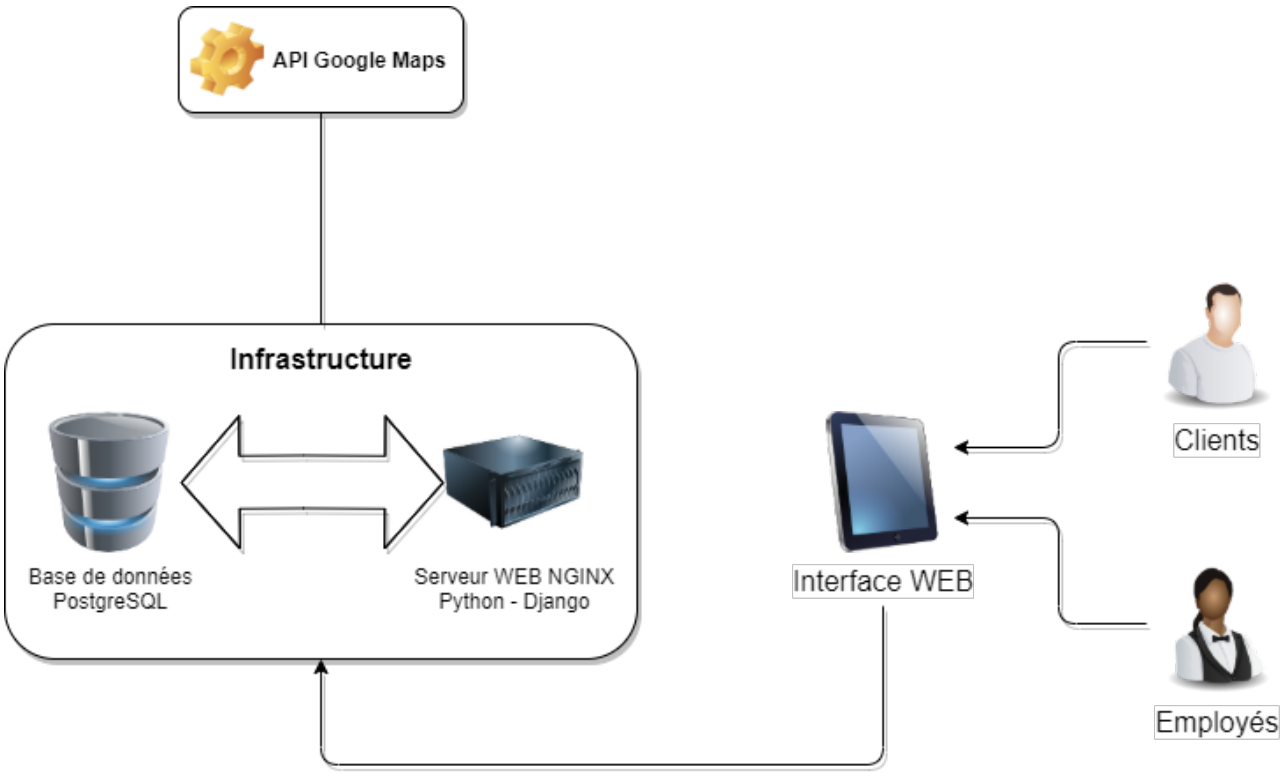
Cette dernière partie met en évidence les composants qui entrent en jeu lors du processus de commande. Nous pouvons voir que pour accéder au panier « **:ShoppingCart** », il est nécessaire que l'utilisateur soit authentifié « **:Authentication** ».

Il existe également une relation avec « **:Order** » se trouvant dans « **OrderManager** ». Celui-ci fournit ensuite les informations au composant « **:Bill** » chargé de la partie facturation et qui fournit, lui-même, les informations de paiement à la banque. Illustré ici par le composant « **:Payment** » dans structure externe « **Bank** ».

3.5 - Application Web

La pile logicielle est la suivante :

- Serveurs **Ubuntu Server 20.04 LTS**
- Application **Django** (*version 3.2 LTS*) / **Python 3.7** / **PostgreSQL 12.6** / **HTML5** / **CSS3** / **Bootstrap4**
- Serveur *HTTP* **NGINX 1.18**



*Solution **OC Pizza***

3.5.1 - Description

L'interface *WEB* étant accessible depuis n'importe quel périphérique (**PC/Mac**, tablette **iOS/Android**, Smartphone **iOS/Android**), il est nécessaire que celle-ci soit « **Responsive** ». C'est-à-dire qu'elle pourra s'adapter à n'importe quel périphérique et quel que soit la résolution.

Cela permettra, dans un premier temps, de s'affranchir du développement d'application pour **iOS** et **Android**. Il sera toujours possible de le faire, si nécessaire, par la suite.

Il sera tout de même judicieux de proposer une version « mobile » du site *web*, plus adaptée aux petites diagonales sur certains smartphones notamment.

Concernant les employés, le pizzaïolo disposera d'une tablette (**iOS** ou **Android**) pour interagir avec l'application *Web*. Le livreur, quant à lui, disposera d'un smartphone (**iOS** ou **Android**) avec abonnement *4G*, nécessaire à l'utilisation de l'application *Web* et notamment la Géolocalisation.

L'application sera hébergée sur un serveur (**Linux**) Web **NGINX** où seront installés **Python** et **Django**. Un autre serveur (**Linux**) où **PostgreSQL** sera installé hébergera la base de données. Le serveur *Web* pourra communiquer avec la base de données lorsque cela sera nécessaire.

4 - ARCHITECTURE DE DEPLOIEMENT

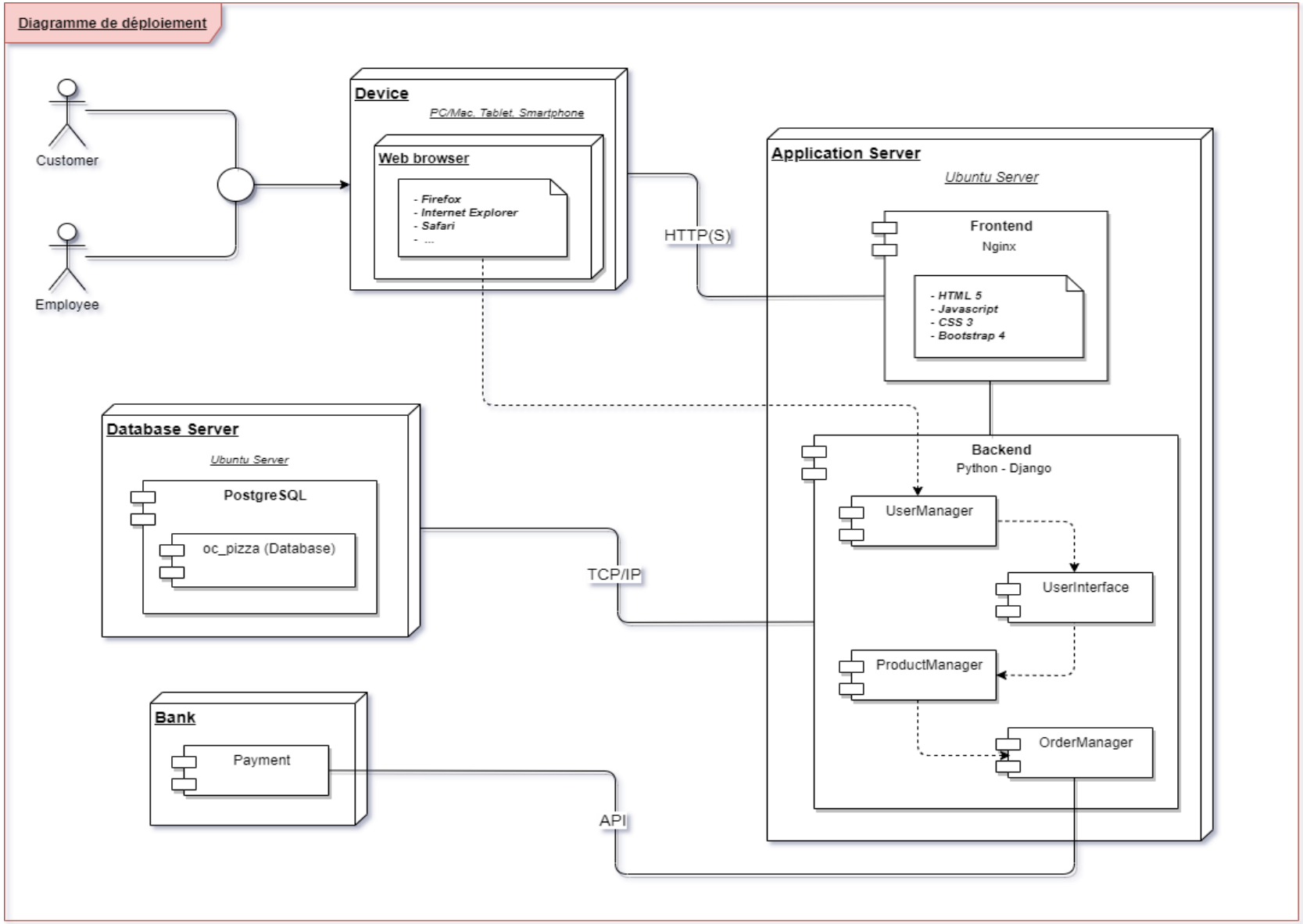


Diagramme de déploiement
Customer
Employee
Device
PC/Mac, Tablet, Smartphone
Web browser
- Firefox
- Internet Explorer
- Safari
- ...
HTTP(S)
Application Server
Ubuntu Server
Frontend
Nginx
- HTML 5
- Javascript
- CSS 3
- Bootstrap 4
Backend
Python - Django
UserManager
UserInterface
ProductManager
OrderManager
Database Server
Ubuntu Server
PostgreSQL
oc_pizza (Database)
TCP/IP
Bank
Payment
API

Ce diagramme illustre la façon dont les éléments (nécessaires au bon fonctionnement de l'application) sont répartis et communiquent au sein de l'infrastructure.

A noter que le système d'exploitation installé sur les différents serveurs sera ***Ubuntu Server****, basé sur* ***Debian****.*

4.1 - Utilisateurs :

Nous pouvons voir les deux types d'utilisateurs (**Customer** et **Employee**), qui, pour interagir avec l'interface web, utilisent un périphérique (*PC/Mac*, tablette ou smartphone, l'application étant *responsive*). Par le biais d'un navigateur *web* (*Firefox*, *Internet Explorer*, *Safari*, *Chrome* ...), ils communiquent avec le serveur *web* nommé ici « **Web Server** » en utilisant le protocole **HTTP(S)**.

4.2 - Serveur d'application (Frontend) :

Le serveur d'application (**Application Server**) héberge donc, comme son nom l'indique, l'application « **OC Pizza** ». **Python**, ainsi que le Framework **Django** sont installés sur ce serveur. Nous pouvons diviser l'application en deux groupes, le « **Frontend** » et le « **Backend** ».

Le « **Frontend** », peut également être défini comme étant le serveur **HTTP**. En tant que serveur **HTTP**, nous utilisons le logiciel « **NGINX** » pour traiter, notamment, les requêtes **HTTP** que le serveur réceptionnera.

Concernant la « partie visible », nous utilisons **HTML 5**, **Javascript**, **CSS3** et **Bootstrap4** pour créer l'interface graphique de l'application.

4.3 - Serveur d'application (Backend) :

La partie « **Backend** » est la partie immergée, le cerveau de l'application. Elle permet, notamment, d'interagir avec la base de données par l'intermédiaire de l'**ORM** intégré à **Django** et de renvoyer les informations au « **Frontend** » qui se chargera de les afficher sur le navigateur de l'utilisateur.

Concernant les relations, nous pouvons voir que l'utilisateur interagit avec l'élément « **UserManager** » notamment lors de l'authentification. Lui-même est lié à l'élément « **UserInterface** » qui est chargé d'afficher l'interface de l'utilisateur authentifié.

On peut également noter la liaison entre « **OrderManager** » et « **Payment** » nécessaire lors du paiement d'une commande à la banque. Cet élément étant externe à l'infrastructure, il est nécessaire d'utiliser une *API*.

Ce serveur communique également avec le serveur de base de données « **Database Server** » en utilisant le protocole **TCP/IP**

4.4 - Serveur de Base de données :

Nous terminons avec le serveur hébergeant la base de données. Ce serveur communique uniquement avec le serveur d'application.

PostgreSQL 12.6 est installé sur ce serveur ainsi que **PgAdmin 4** pour faciliter l'administration de la base hébergée et nommée « **oc_pizza** », qui contient elle-même toutes les tables nécessaires au fonctionnement de l'application.

5 - ARCHITECTURE LOGICIELLE

5.1 - Principes généraux

Les sources et versions du projet sont gérées par **Git**, alors que les dépendances de l'application seront gérées par le gestionnaire de paquet « **pip** » concernant la partie **Django**.

Django est un **Framework Python** basé sur une architecture **MVT** (**Model** – **View** – **Template**). Il permet notamment d'interagir avec la base de données par l'intermédiaire de son **ORM**, nous parlerons ici du « **Model** ». Il peut également recevoir des requêtes **HTTP** et d'y répondre en renvoyant (**View**), par exemple, un **Template** qui est en fait un fichier **HTML** pouvant recevoir des objets **Python**.

5.1.1 - Les couches

L'architecture applicative est la suivante :

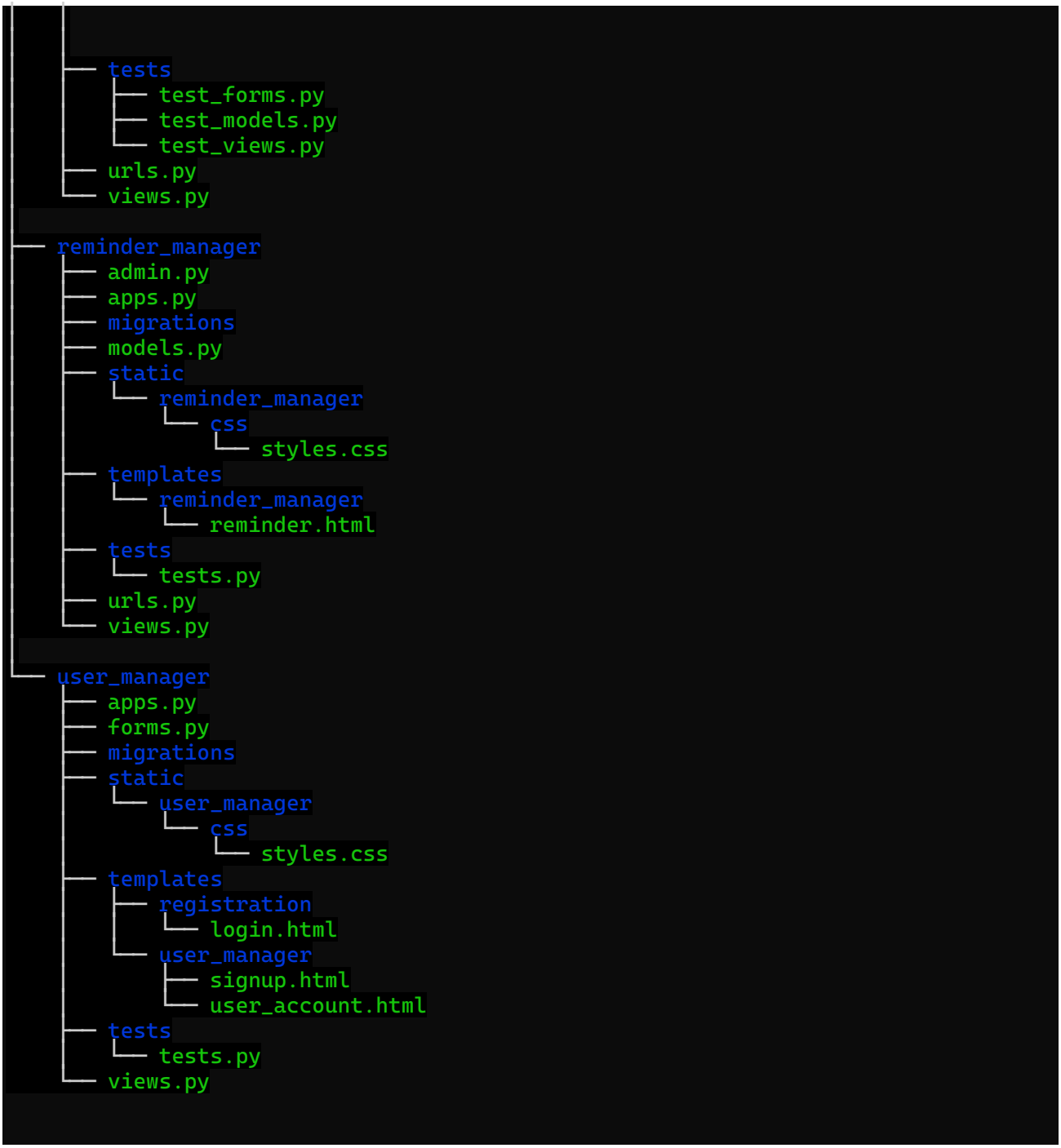
- Une couche **Business** : Responsable de la logique métier du composant
- Une couche **View** : Responsable de la logique de l'application
- Une couche **Model** : Responsable de la représentation des objets métiers
- Une couche **Template** : Responsable des interfaces « utilisateur » de l'application

5.1.2 - Structure des sources

La structuration des répertoires du projet suit la logique suivante :

– Les répertoires sources sont créés de façon à respecter la philosophie de **Django** (à savoir : « **Model** – **View** – **Template** »)

```
OC_PIZZA_APP
├── manage.py
├── oc_pizza_app
│   ├── asgi.py
│   ├── settings
│   │   ├── production.py
│   │   └── travis.py
│   ├── staticfiles
│   │   └── ...
│   ├── urls.py
│   └── wsgi.py
│
├── order_manager
│   ├── apps.py
│   ├── migrations
│   │   └── ...
│   ├── templates
│   │   └── order_manager
│   │       └── order.html
│   ├── tests
│   │   └── tests.py
│   └── views.py
│
├── product_manager
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── management
│   │   └── commands
│   │       └── ...
│   ├── migrations
│   │   └── ...
│   ├── models.py
│   ├── static
│   │   └── product_manager
│   │       ├── css
│   │       │   └── styles.css
│   │       └── js
│   │           └── scripts.js
│   ├── templates
│   │   ├── 404.html
│   │   └── product_manager
│   │       ├── base.html
│   │       ├── base_product.html
│   │       ├── detail.html
│   │       ├── index.html
│   │       ├── log_users.html
│   │       ├── mentions.html
│   │       ├── result.html
│   │       └── search.html
```


tests
test_forms.py
test_models.py
test_views.py
urls.py
views.py
reminder_manager
admin.py
apps.py
migrations
models.py
static
reminder_manager
css
styles.css
templates
reminder_manager
reminder.html
tests
tests.py
urls.py
views.py
user_manager
apps.py
forms.py
migrations
static
user_manager
css
styles.css
templates
registration
login.html
user_manager
signup.html
user_account.html
tests
tests.py
views.py

6 - POINTS PARTICULIERS

6.1 - Gestion des logs

La gestion des *erreurs* (et des *exceptions*) sera assurée par **Sentry**. Nous l'utiliserons également en tant que gestionnaire de **Logs** pour l'application **OC Pizza**. En plus de la gestion des erreurs et des logs, il sera relié au **Repository GitHub** de l'application afin de pouvoir suivre les issues éventuelles.

Nous utiliserons **NewRelic** pour surveiller les performances de l'application et, éventuellement l'optimiser. Le monitoring fourni par l'hébergeur (**DigitalOcean**) permettra également de surveiller les performances « matériels » du serveur. Afin, par exemple, de nous assurer que le serveur soit correctement dimensionné pour l'application.

6.2 - Fichiers de configuration

6.2.1 - Application OC_Pizza

```
OC_PIZZA_APP
├── manage.py
├── oc_pizza_app
│   ├── asgi.py
│   ├── settings
│   │   ├── production.py
│   │   └── travis.py
```

Le fichier « *production.py* » contient la configuration (finale) de l'application. Nous retrouvons dans ce fichier :

- La configuration de **Sentry** et notamment le « *niveau* » des alertes qui seront envoyées.
- Les hôtes autorisés (*ALLOWED_HOSTS*), il s'agira ici de saisir le nom de domaine attribué à l'application.
- Les informations concernant le serveur de base de données **PostGres** ainsi que la base de données elle-même.

Un autre fichier nommé « *travis.py* » contient lui la configuration vers l'outil d'intégration continue nommée « **Travis CI** ». Celui-ci contiendra un fichier de configuration simpliste de l'application nécessaire à la bonne exécution de l'outil, notamment concernant la base de données de données **Postgres**.

6.2.2 - Supervisor

```
/etc/supervisor
├── conf.d
│   └── oc_pizza_supervisor.conf
```

Supervisor est un gestionnaire de services, il permet donc de lancer des services au démarrage du serveur et de les surveiller ensuite. S'ils échouent pour une raison ou une autre, il se chargera de les redémarrer.

6.2.3 - NGINX

```
/etc/nginx
├── sites-available/
│   ├── default
│   └── oc_pizza
```

NGINX est un serveur **HTTP** (ainsi qu'un « *reverse proxy* ») couramment utilisé, il est même, à priori, le plus utilisé au monde depuis 2019. Son rôle est d'interpréter les requêtes émises par les navigateurs et de les renvoyer vers l'application.

Le fichier de configuration de l'application contient le nom de domaine, la localisation des fichiers statiques, ou encore, la configuration du *proxy*.

6.3 - Environnement de développement

Concernant le développement de l'application ou l'ajout de nouvelles fonctionnalités, il est recommandé de le faire au sein d'un « *environnement virtuel* ». Cela permettra alors d'intégrer les dépendances de cette application (et d'en ajouter si nécessaire) sans pour autant le polluer par des autres éléments utilisés dans cadre différent. Les dépendances actuelles sont répertoriées dans un fichier « *requirements.txt* » présent à la racine du projet.

6.4 - Procédure de packaging / livraison

Le développement de l'application respecte les valeurs de la méthode **AGILE**. Dans cette optique, les fonctionnalités sont ajoutées de façon continue et incrémentielle.

Pour cela, et comme indiqué plus haut, nous utilisons un outil d'intégration continue nommé **Travis CI**. Une fois l'application développée, le déroulement se fait de la façon suivante :

- Création d'un « *pull request* »
- **Travis CI** prépare son environnement et se charge ensuite d'exécuter les différents tests unitaires, d'intégration et fonctionnels
- Si un ou plusieurs tests échouent, une notification indique aux développeurs que la fonctionnalité ne peut pas être intégrée.
- Si les tests passent, le service indique que la fonctionnalité peut être intégrée au à l'application

7 - GLOSSAIRE

B

Backend
Partie de l'application en arrière plan, pas directement accessible par un utilisateur 16

E

environnement virtuel
Un environnement virtuel est un répertoire contenant une installation autonome (indépendante de celle du système d'exploitation). 22

F

Framework
Ensemble d'outils et de composants logiciels à la base d'une application 16, 18
Frontend
Partie d'une application responsable de l'interface utilisateur 16

H

HTTP
Protocole de transmission permettant d'accéder à des pages web 12, 16, 18, 22

I

intégration continue
Ensemble de pratiques utilisées consistant à vérifier à chaque modification du code source que le résultat des modifications ne produit pas de régression dans l'application développée 22, 23

L

Logs
Type de fichier dont la mission consiste à stocker un historique des événements 21

M

monitoring

O

P

R

T