



# **OC PIZZA**

## **Application web de vente en ligne et logistique**

### **Dossier de conception technique**

### **VERSION 1.0**



**Auteur**

**Mélanie OBRINGER**

*Analyste-Programmeuse*

# TABLE DES MATIERES

<b>1 - VERSIONS.....</b>	<b>2</b>
<b>2 - INTRODUCTION.....</b>	<b>3</b>
<b>2.1 - OBJET DU DOCUMENT .....</b>	<b>3</b>
<b>2.2 - REFERENCES .....</b>	<b>3</b>
<b>3 - ARCHITECTURE TECHNIQUE .....</b>	<b>4</b>
<b>3.1 - APPLICATION OC PIZZA.....</b>	<b>4</b>
<b>3.2 - LA BASE DE DONNEES .....</b>	<b>4</b>
3.2.1 - SGBD UTILISE .....	4
3.2.2 - MPD.....	4
3.2.3 - PRESENTATION DES TABLES .....	5
TABLE « ACHETEUR ».....	5
TABLE « CLIENTWEB » .....	5
TABLE « EMPLOYE » .....	6
TABLE « ADRESSE » .....	6
TABLE « POINTDEVENTE » .....	6
TABLE « COMMANDE » .....	6
TABLE « LIGNECOMMANDE » .....	7
TABLE « ARTICLE ».....	7
TABLE « INGREDIENT » .....	7
TABLE « ARTICLEINGREDIENT » .....	7
TABLE « STOCK » .....	7
<b>4 - ARCHITECTURE DE DEPLOIEMENT .....</b>	<b>8</b>
<b>4.1 - DIAGRAMME DE DEPLOIEMENT ET DESCRIPTION .....</b>	<b>8</b>
<b>4.2 - SERVEUR DE DEPLOIEMENT .....</b>	<b>9</b>
<b>5 - ARCHITECTURE LOGICIELLE .....</b>	<b>10</b>
<b>5.1 - PRINCIPES GENERAUX .....</b>	<b>10</b>
5.1.1 - LES COUCHES.....	10
5.1.2 - LES MODULES .....	10
5.1.3 - LE DIAGRAMME DE COMPOSANTS.....	10
5.1.4 - STRUCTURE DES SOURCES.....	12
<b>6 - POINTS PARTICULIERS.....</b>	<b>13</b>
<b>6.1. - LA GESTION DES LOGS .....</b>	<b>13</b>
<b>6.2. - RESSOURCES.....</b>	<b>14</b>
6.2.1 - API GOOGLE MAPS.....	14
6.2.2 - IMAGES.....	14
6.2.3 - BASE DE DONNEES.....	14
6.2.4 - ENVIRONNEMENT DE DEVELOPPEMENT .....	14
<b>6.3 - PROCEDURE DE PACKAGING ET DE LIVRAISON .....</b>	<b>14</b>
<b>7 - GLOSSAIRE .....</b>	<b>15</b>

# 1 - VERSIONS

AUTEUR	DATE	DESCRIPTION	VERSION
Mélanie OBRINGER	01/09/2019	Création du document	1.0

## 2 - INTRODUCTION

### 2.1 - Objet du document

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

L'objectif du document est d'informer et aider les développeurs pour la conception, la maintenance et l'évolution de l'application.

Les éléments du présent dossier découlent :

- du domaine fonctionnel
- des spécifications techniques
- du modèle physique de donnée

### 2.2 -Références

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

Projet 8 – Dossier_d_exploitation.pdf – 1.0	Dossier d'exploitation de l'application
Projet 8 – Dossier_de_conception_fonctionnelle.pdf – 1.0	Dossier de conception fonctionnelle de l'application

## 3 - ARCHITECTURE TECHNIQUE

Afin de répondre au mieux aux besoins du client, nous avons découpé l'application en deux parties, l'application front-end et back-end.

### 3.1 - Application OC Pizza

**Le front-end est la partie visible du site-web** (partie où il y a les interactions client).

Il sera réalisé à l'aide des langages de programmation *HTML*, *CSS* et *JavaScript* car ce sont de véritables standards qui sont la base de tout projet de développement web.

**Le back-end est la partie logique du site-web.** Il sera réalisé avec le langage de programmation *Python*, jumelé à son *Framework Django*. Cette interface back-end communiquera avec la base de données *PostgreSQL*.

Python est un langage polyvalent, multiplateforme et open source. Le système est fiable, très utilisé, un grand standard de l'industrie (Google, Instagram ou encore Dropbox ont été développés en Python).

Django permet de séparer des données de la vue et du contrôleur, sa conception devient plus claire et efficace, car cela permet un gain de temps de maintenance et d'évolution du site.

### 3.2 - La base de données

La base de données permet de stocker et de retrouver l'intégralité des données et informations en rapport avec le groupe de pizzerias.

#### 3.2.1 - SGBD utilisé

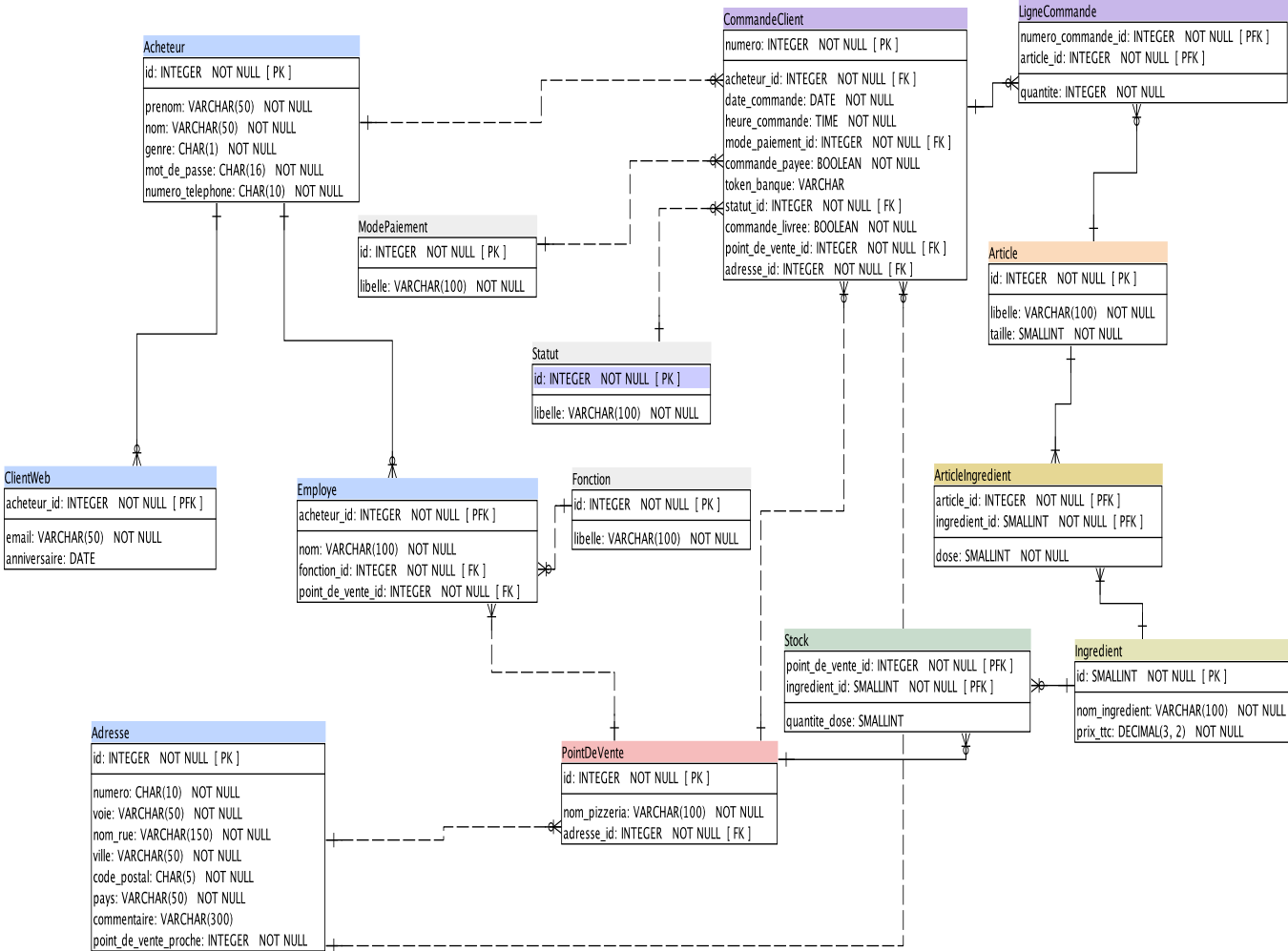
Un Système de Gestion de Base de Données (SGBD) est un logiciel qui permet le stockage d'informations dans une base de données. Il permet de créer, lire, mettre à jour, et supprimer les données qui sont contenues dans la base de données. Ces données seront stockées dans le SGBD **PostgreSQL**, qui est une **base de données open source, populaire et robuste**.

#### 3.2.2 - MPD

A partir du diagramme de classes présenté dans le dossier de conception fonctionnelle, nous avons élaboré le MPD (Modèle Physique de Données).

Il permet d'avoir une **représentation graphique** de la structure d'une base de données et de mieux **comprendre les relations entre les différentes tables**.

Les parties suivantes auront pour but de détailler les **types utilisés pour chaque attribut de table ainsi que les clés étrangères et primaires**.



### 3.2.3 - Présentation des tables

### Table « Acheteur »

Cette table regroupe les informations de tous les « Acheteurs », en général.

Toutes les colonnes de cette table sont renseignées (NOT NULL).

La clé primaire est **id**, avec un **AUTO INCREMENT**, et de type **INTEGER**.

Pour les autres colonnes, le type est :

- **VARCHAR**, avec **50** caractères pour le **nom** et le **prenom** ;
- **CHAR**, avec **10** caractères pour **numero\_telephone**, **1** pour **genre** (« H » ou « F »), et **16** caractères pour le **mot de passe**.

Table « ClientWeb »

Cette table regroupe les informations spécifiques d'une **spécialisation** des « Acheteurs » que sont les « Clients Internet ».

Les colonnes **email (VARCHAR, 100)** et **anniversaire (DATE)** doivent être obligatoirement renseignées (**NOT NULL**).

La colonne **acheteur\_id** est une clé étrangère se référant à l'**id** correspondant dans la table **Acheteur**.

### Table « Employe »

Cette table regroupe les informations spécifiques d'une **spécialisation** des « Acheteurs » que sont les employés d'OC Pizza.

Ici, **toutes les colonnes sont à renseigner (NOT NULL)**.

Le **nom** et la **fonction** sont un **VARCHAR(50)**.

La colonne **acheteur\_id** est une **clé étrangère** se référant à l'id correspondant dans la table **Acheteur**.

### Table « Adresse »

**Les colonnes numero, voie, nom\_rue, ville, code\_postal, pays et point\_de\_vente\_proche doivent être renseignées (NOT NULL).**

La colonne **commentaire** n'est pas à renseigner obligatoirement. Elle permet d'ajouter des précisions par rapport à l'adresse (étage, code...).

La **clé primaire** est **id**, avec un **AUTO\_INCREMENT**, et de type **INTEGER**.

Pour les colonnes, le type de données est :

- **VARCHAR**, avec **150** caractères pour **nom\_rue** ;
- **VARCHAR**, avec **50** caractères pour **voie, ville, pays** ;
- **CHAR**, avec **10** caractères pour **numero** et **5** pour **code\_postal** ;
- **VARCHAR**, avec **300** caractères pour **commentaire** ;

### Table « PointDeVente »

La colonne **nom**, de type **VARCHAR**, avec **100** caractères, doit être renseignée (**NOT NULL**).

La **clé primaire** est **id**, avec un **AUTO\_INCREMENT**, de type **INTEGER**.

La colonne **adresse\_id** est une **clé étrangère** se référant à l'id correspondant dans la table **Adresse**, afin de récupérer l'adresse de chaque pizzeria.

### Table « Commande »

**Toutes les colonnes, à part bank\_token** (car le paiement n'est pas forcément par carte bancaire), doivent être renseignées (**NOT NULL**).

La **clé primaire** est **numero**, avec un **AUTO\_INCREMENT**, et de type **INTEGER**.

Pour les colonnes, le type de données est :

- **DATE**, pour **date\_commande** ;
- **TIME**, pour **heure\_commande** ;
- **VARCHAR**, pour **statut** ;
- **BOOLEAN**, pour **commande\_livree** (true ou false si la commande est à livrer à domicile ou non) ;
- **VARCHAR**, pour **mode\_paiement** ;
- **BOOLEAN**, pour **commande\_payee** (true ou false si le paiement a été effectué ou non) ;
- **VARCHAR**, avec pour **bank\_token**.

Il y a **3 clés étrangères** :

- la colonne **acheteur\_id** se référant à l'id correspondant dans la table **Acheteur**, afin de relier chaque commande à un « acheteur ».
- la colonne **point\_de\_vente\_id** se référant à l'id correspondant dans la table **PointDeVente**, afin de relier chaque commande à une pizzeria du groupe. Le choix de la pizzeria dépend du résultat du calcul de géolocalisation entre le lieu de livraison et les différentes pizzerias : la plus proche sera choisie.
- la colonne **adresse\_id** se référant à l'id correspondant dans la table **Adresse**, afin de relier chaque commande à l'adresse du client.

### Table « LigneCommande »

La table **LigneCommande** est une **table d'association** (association many-to-many).

La colonne **quantite**, de type **INTEGER**, doit être renseignée (**NOT NULL**). C'est la quantité de chaque type d'article commandé.

Il y a 2 clés étrangères :

- la colonne **numero\_commande** de la table **Commande**, afin de relier chaque ligne de commande à une commande.
- la colonne **article\_id** se référant à l'**id** de la table **Article**, afin de relier chaque ligne de commande à un article.

### Table « Article »

La colonne **libelle**, de type **VARCHAR**, avec **100** caractères, doit être renseignée (**NOT NULL**).

La colonne **taille**, de type **SMALLINT**, doit être renseignée (**NOT NULL**). Elle prend la valeur **1** pour les articles de taille « Format standard » et **2** pour les articles de taille « Grand Format ».

La clé primaire est **id**, avec un **AUTO\_INCREMENT**, et de type **INTEGER**.

### Table « Ingredient »

La colonne **nom\_ingredient**, de type **VARCHAR**, avec **100** caractères, doit être renseignée (**NOT NULL**).

La colonne **prix\_ttc**, de type **DECIMAL**, de **3 chiffres dont 2 après la virgule**, doit être renseignée (**NOT NULL**).

La clé primaire est **id**, avec un **AUTO\_INCREMENT**, et de type **INTEGER**.

### Table « ArticleIngredient »

La table **ArticleIngredient**, n'apparaît pas dans le Diagramme de classes. C'est une **table d'association** qui doit être créée du fait de l'association many-to-many entre les tables **Article** et **Ingredient**.

La colonne **dose**, de type **SMALLINT**, doit être renseignée (**NOT NULL**) et prend la valeur **1** pour les articles « Format normal », et **2** pour les articles « Grand Format ». C'est le nombre de dose de chaque ingrédient que nécessite chaque article.

Il y a 2 clés étrangères :

- la colonne **article\_id** se référant à l'**id** correspondant dans la table **Article**, afin de relier chaque ingrédient à un article donné.
- la colonne **ingredient\_id** se référant à l'**id** correspondant dans la table **Ingredient**, afin de relier chaque article à un ingrédient donné.

### Table « Stock »

La table **Stock** est une table d'association (association many-to-many).

La colonne **quantite\_dose**, de type **SMALLINT**, doit être renseignée mais peut être de valeur nulle (fin de stock d'un ingrédient). C'est la quantité de doses que chaque pizzeria possède de chaque ingrédient.

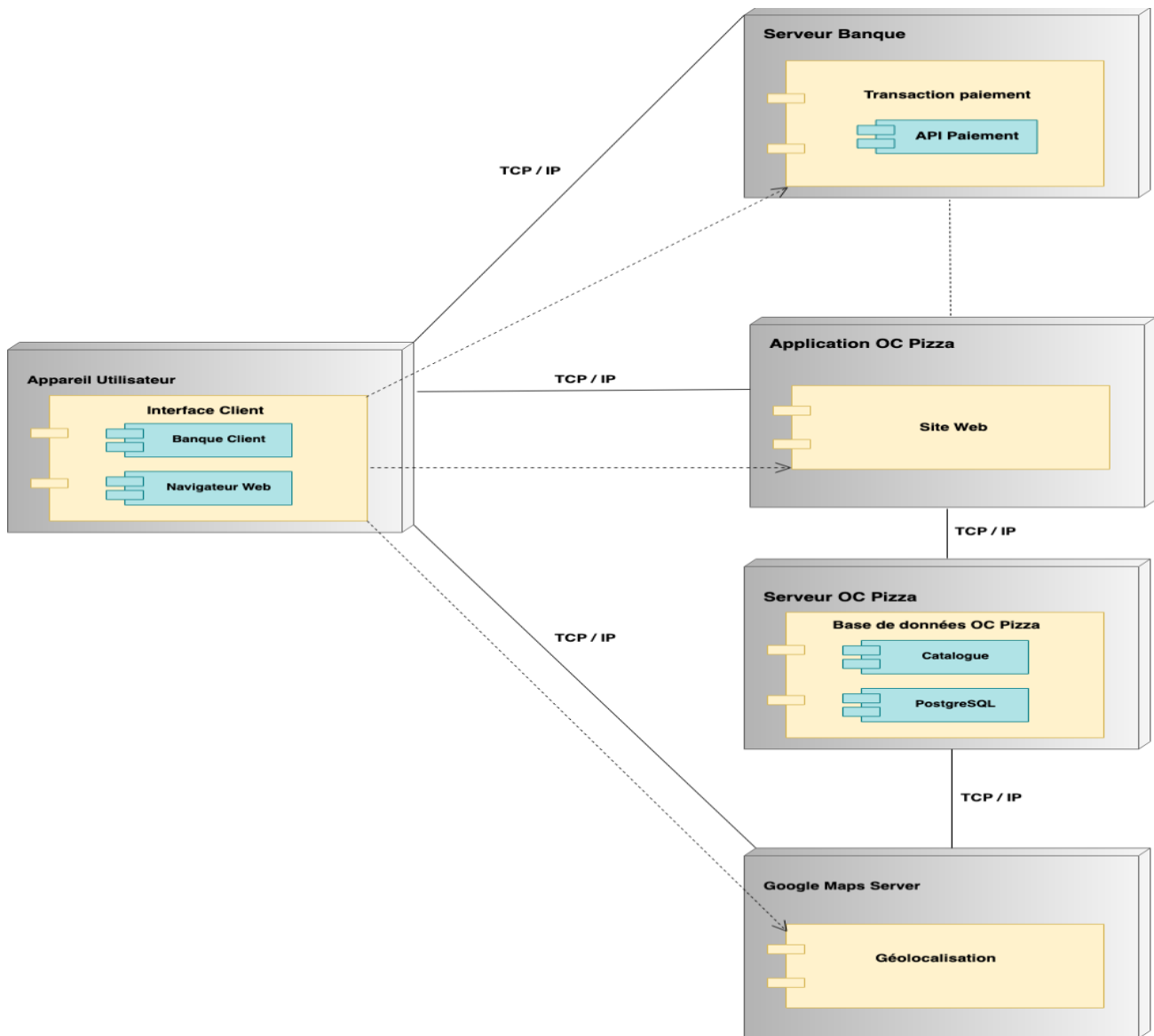
Il y a 2 clés étrangères :

- la colonne **point\_de\_vente\_id** se référant à l'**id** correspondant dans la table **PointDeVente**, afin de relier chaque pizzeria donnée à un stock d'ingrédient.
- la colonne **ingredient\_id** se référant à l'**id** correspondant dans la table **Ingredient**, afin de relier un stock d'ingrédient donné à chaque pizzeria.



## 4 - ARCHITECTURE DE DEPLOIEMENT

### 4.1 - Diagramme de déploiement et description



Comme on peut le voir dans le diagramme, il y a cinq nœuds qui représentent les éléments hardware ou software faisant partie du système.

-> **Appareil Utilisateur** indique l'ordinateur avec lequel l'utilisateur se connecte.

L'**Interface Client** représente l'interface utilisateur, à travers laquelle il peut accéder au site web d'OC Pizza (via le **Navigateur Web**) et payer en ligne (via la banque **Banque Client**).

-> **Serveur Banque** indique le serveur qui gère les requêtes pour les paiements en ligne, effectués grâce aux APIs des banques (**API Paiement**).

Le transfert des données entre les deux nœuds se fait grâce au protocole TCP/IP. La dépendance entre les composants **Interface Client** et **Transaction Paiement** est marquée par la flèche en pointillés.

-> **Application OC Pizza** indique l'ensemble de pages web qui composent le site web Oc Pizza, comme décrit par le composant **Site Web**.

Le transfert de données entre ce nœud et le nœud **Appareil Utilisateur** se fait grâce au protocole TCP/IP.

La dépendance entre les composants **Interface Client** et **Site Web** est marquée par la flèche en pointillés.

-> **Serveur OC Pizza** indique le serveur chargé de gérer les requêtes des pages du site web.

Il repère les informations grâce à la base de données dédiée **Base de données OC Pizza** et les affiche à l'utilisateur sous forme de **Catalogue**.

Les interactions avec la base de données sont assurées par le SGBDR **PostgreSQL**.

L'échange des données entre les nœuds **Application OC Pizza** et **Serveur OC Pizza** se fait toujours avec TCP/IP.

-> **Serveur Google Maps** indique le serveur Google chargé de gérer les requêtes de géolocalisation.

Le transfert des données entre ce nœud et **Serveur OC Pizza** est réalisé encore à l'aide du protocole TCP/IP.

Le **Serveur Google Maps** est relié via TCP/IP aussi au nœud **Appareil Utilisateur** et il y a une relation de dépendance entre les composants **Interface Client** et **Géolocalisation**.

## 4.2 - Serveur de déploiement

L'application OC PIZZA sera déployée sur le serveur OVH, entreprise française implantée à l'étranger proposant des temps de latence réduits et un réseau ultra-sécurisé.

L'offre d'hébergement Pro (1 nom de domaine, 250 Go d'espace disque, 100 adresses e-mails, Trafic illimité) dont le prix est de 7,19 euros TTC par mois proposée par OVH est une solution adaptée à la taille du projet.

## 5 - ARCHITECTURE LOGICIELLE

### 5.1 - Principes généraux

Les sources et versions du projet sont gérées par Git et les dépendances par PyPi.

Le développement de l'application se présente sous la forme d'un projet Django, divisé en deux parties :

- Ventes
- Administration Restaurant

#### 5.1.1 - Les couches

L'architecture applicative est la suivante :

- Une couche **model** : implémentation du modèle des objets métiers
- Une couche **template** : représente les gabarits des données renvoyées
- Une couche **vue** : donne accès au modèle et au template adapté à la requête

Django utilise une architecture (design-pattern) proche de MVC : le MVT. MVT signifiant . **Modèle - Vue –Template**  
Le **modèle** permet de faire de **requêtes à la base de données**. La **vue reçoit des requêtes HTTP** et va **lire dans la base** l'élément correspondant, **pour générer le Template**. Le **Template est un fichier HTML** qui **reçoit des objets Python** et qui est **lié à une vue**.

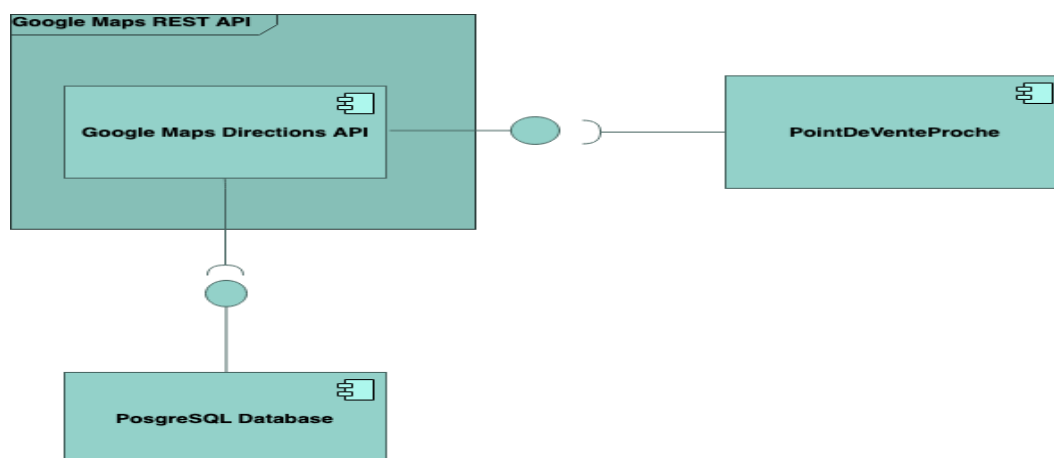
#### 5.1.2 - Les modules

Django-Cron : Permet de créer des tâches « Cron » pouvant s'exécuter à intermédiaire régulier, comme une mise à jour de la base de données vis-à-vis d'une source externe.

#### 5.1.3 - Le diagramme de composants

Les Diagrammes de composants (Component Diagrams) décrivent l'organisation du système du point de vue des éléments logiciels. Ils mettent en évidence les dépendances entre les composants, et décrivent ici les interfaces entre les composants internes du système OC Pizza et les composants externes (Banque, Google Maps).

### LA GEOLOCALISATION (DIAGRAMME ET DESCRIPTION)



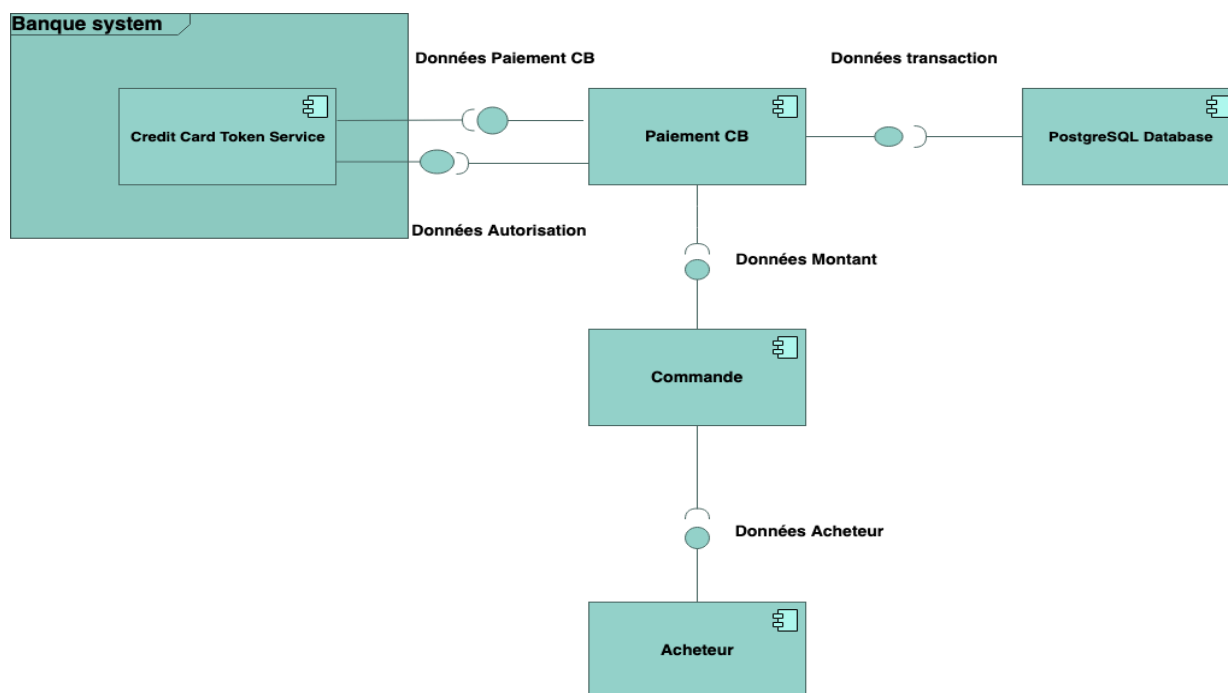
Le composant PostgreSQL Database envoie à l'API REST Google Maps Directions les informations sur l'adresse de livraison d'un client et l'adresse des différentes pizzerias du groupe OC Pizza.

Ce composant externe **calcul le temps de parcours** et l'itinéraire à privilégier, pour un véhicule, entre cette adresse et celles des points de vente et **envoie l'information** à l'interface requise du **composant PointDeVenteProche** qui va **retenir la pizzeria la plus proche du lieu de livraison**.

Cette information (la pizzeria la plus proche) sera **stockée dans la base de données** dans la colonne **point\_de\_vente\_proche de la table Adresse** pour l'adresse de livraison donnée.

Documentation : <https://developers.google.com/maps/documentation/directions/start?hl=fr>

## LE PAIEMENT EN LIGNE (DIAGRAMME ET DESCRIPTION)



Le **composant Acheteur** envoie au **composant Commande** les informations sur la commande et celui-ci **calcule le montant total**. Ce montant est requis par le **composant Credit Card Payment** qui récupère aussi le nom du client.

Ce composant va **échanger** avec le système bancaire via le **composant Credit Card Token Service** des informations sur le client et le montant à régler. Le client aura directement accès au service de carte bancaire de la banque pour rentrer ses coordonnées personnelles (numéro de carte, date d'expiration...).

En retour, **ce composant externe renvoie un token** au composant interne avec **l'information cryptée sur la réalisation du paiement**.

Cette information sera **gardée en base de données**, via le composant PostgreSQL Database, le système d'OC Pizza.

Exemples d'API proposées au client autres que sa banque : **Mangopay** et **Lemonway**. Les 2 services proposent un paiement via CB, visa, Mastercard et de nombreux autres moyens de paiement.

**Le client a opté pour Monetico du Crédit Mutuel, sa banque.**

### 5.1.4 - Structure des sources

La structuration des répertoires du projet suit la logique suivante :

- Les répertoires sources sont créés selon la nomenclature du Framework Django

```
Projet_OC_PIZZA/  
├── docs  
├── requirements.txt  
├── README.md  
├── OC_PIZZA/  
│   ├── manage.py  
│   ├── settings/  
│   │   ├── __init__.py  
│   │   └── settings.py  
│   ├── urls.py  
│   └── wsgi.py  
├── ventes/  
│   ├── admin.py  
│   ├── apps.py  
│   ├── __init__.py  
│   ├── migrations/  
│   │   └── __init__.py  
│   ├── models.py  
│   ├── views.py  
│   └── tests.py  
├── administration_restaurants/  
│   ├── admin.py  
│   ├── apps.py  
│   ├── __init__.py  
│   ├── migrations/  
│   │   └── __init__.py  
│   ├── models.py  
│   ├── views.py  
│   └── tests.py  
├── static/  
│   ├── style.js  
│   ├── images/  
│   └── style.css  
└── templates/
```

- `requirements` maintient une liste des dépendances qui doivent être installées avec `pip install`
- `manage.py` est un script qui aide à gérer ou maintenir le site
- `settings.py` contient la configuration du site web
- `__init__.py` permet de déclarer un dossier comme un package importable
- `urls.py` contient une liste de patterns d'urls utilisés
- `admin.py` permet d'ajouter, modifier ou supprimer des données
- `wsgi.py` norme qui sert à définir comment un serveur Python et son application peuvent communiquer
- `models.py` permet de définir les objets
- `views.py` contient les vues
- `static` contient les fichiers html, css, javascript et les images

## 6 - POINTS PARTICULIERS

### 6.1. - La gestion des logs

Pour accéder aux statistiques du site, aller sur :

<https://logs.ovh.net/ocpizza>

La connexion se fait via l'identifiant OVH et le mot de passe associé.

Cliquez sur le lien généré automatiquement dans le **Manager** pour accéder aux statistiques et aux logs.

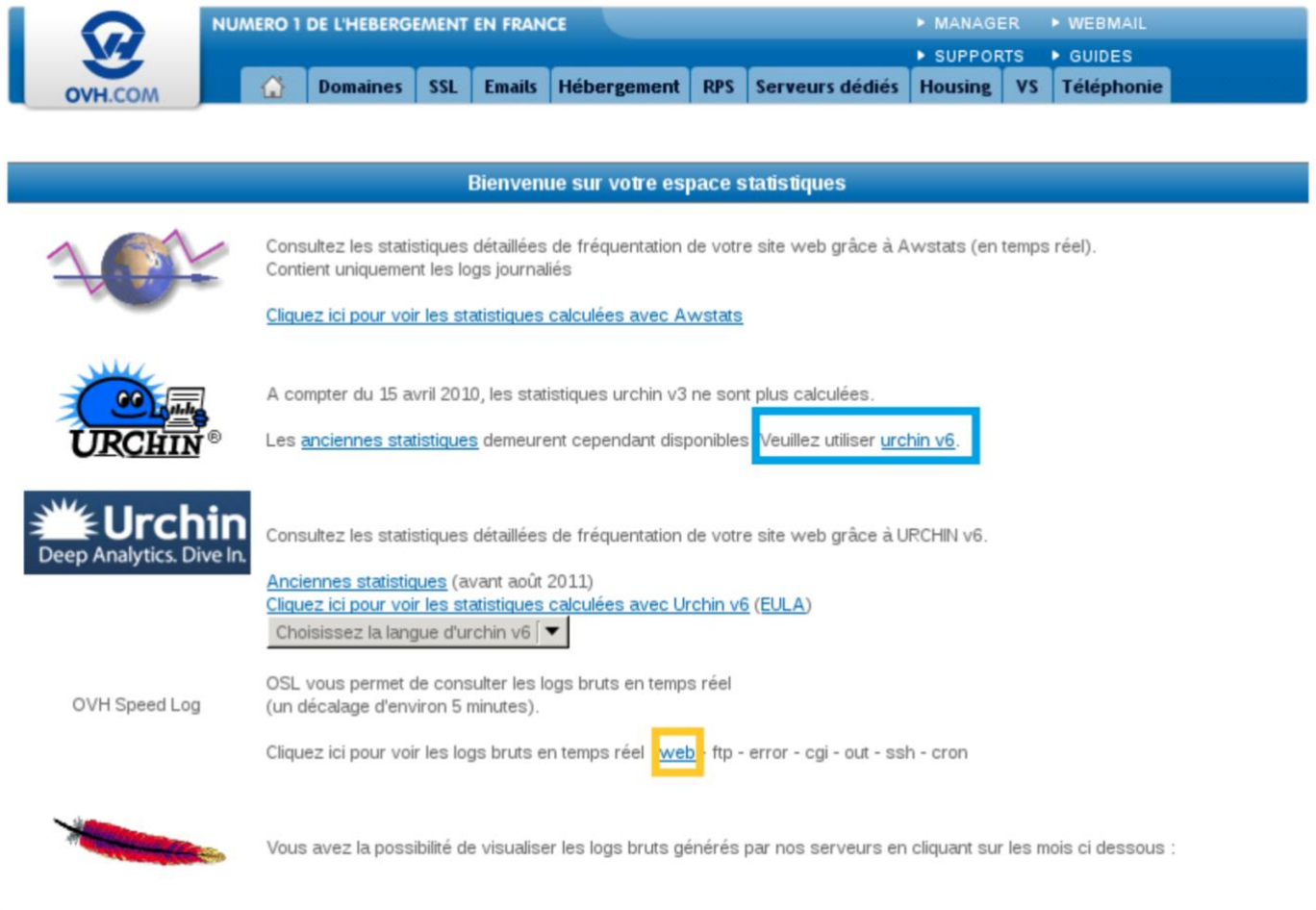
Il faudra s'identifier avec la référence client (Nic-handle) et le mot de passe.

#### Espace statistiques

Une fois connecté à l'espace de statistiques, voici la page qui apparaît (cf capture ci-contre).

Deux possibilités :

- Accéder aux statistiques du site via urchin v6.
- Consulter les logs bruts en temps réel ou sur une période antérieure



The screenshot shows the OVH statistics dashboard. At the top, there's a navigation bar with the OVH logo and various service links like Domains, SSL, Emails, Hébergement, RPS, Serveurs dédiés, Housing, VS, and Téléphonie. Below this, a blue banner reads "Bienvenue sur votre espace statistiques". The main content area is divided into several sections:

- Awstats:** A section with a globe icon and a line graph. It says "Consultez les statistiques détaillées de fréquentation de votre site web grâce à Awstats (en temps réel). Contient uniquement les logs journalisés." and provides a link "Cliquez ici pour voir les statistiques calculées avec Awstats".
- Urchin v3:** A section with the Urchin logo. It states "A compter du 15 avril 2010, les statistiques urchin v3 ne sont plus calculées." and "Les [anciennes statistiques](#) demeurent cependant disponibles. Veuillez utiliser [urchin v6](#)." The text "anciennes statistiques" and "urchin v6" are highlighted with blue boxes.
- Urchin v6:** A section with the Urchin logo and the text "Deep Analytics. Dive In." It says "Consultez les statistiques détaillées de fréquentation de votre site web grâce à URCHIN v6." and provides a link "Cliquez ici pour voir les statistiques calculées avec Urchin v6 (EULA)". Below this is a dropdown menu labeled "Choisissez la langue d'urchin v6".
- OVH Speed Log:** A section with the text "OSL vous permet de consulter les logs bruts en temps réel (un décalage d'environ 5 minutes)." and a link "Cliquez ici pour voir les logs bruts en temps réel". The word "web" in the link is highlighted with a yellow box.
- Feather:** A section with a feather icon. It says "Vous avez la possibilité de visualiser les logs bruts générés par nos serveurs en cliquant sur les mois ci dessous :

## Urchin v6

Ces statistiques donnent des renseignements sur le trafic du site

- Le nombre de visiteurs,
- Le nombre de pages visualisées,
- Le “poids” des pages visualisées,
- Le nombre de requêtes http.
- Les durées moyennes de connexion à l’ensemble de du site ou une page particulière
- Comment les visiteurs du site l’ont-ils connu ?
- Par quels moteurs de recherche ont-ils trouvé l’URL de du site ?
- Quels mots-clés ont-ils utilisé lors de leur recherche ?

## Logs bruts

Il est possible de visualiser les logs du site pratiquement en direct en moins de 15 minutes, ce qui permet de vérifier le bon fonctionnement du site presque en temps réel.

Différents types de logs sont à disposition :

- Logs Web : trouvez ici les différents logs de consultation du site, ainsi que les différentes actions réalisées à partir du site. Cela permet par exemple de repérer des tentatives de hacks.
- Logs FTP : les différentes connexions FTP seront enregistrées et conservées dans ces logs.
- Logs erreur : les différentes erreurs générées par le site.
- Logs CGI : les différents appels aux scripts cgi.bin qui ont été réalisés.
- Logs out : les statistiques de l’hébergement sur les différents appels externes réalisés.
- Logs SSH : ces logs indiquent les différentes connexions réalisées avec le protocole SSH.
- Logs cron : le résultat de l’exécution des tâches planifiées

## 6.2. - Ressources

---

### 6.2.1 - API Google Maps

Pour intégrer l’API Google Maps, se référer à la documentation officielle au lien suivant : <https://developers.google.com/maps/documentation/directions/start?hl=fr>

### 6.2.2 - Images

Le client a fourni les images pour pouvoir livrer l’application opérationnelle.

### 6.2.3 - Base de données

Deux fichiers SQL sont fournis avec les livrables, l’un contient les scripts SQL pour la création de la base de données et les différentes tables sans données, le deuxième contient le dump avec les données pour que le client puisse démarrer l’application de manière opérationnelle.

### 6.2.4 - Environnement de développement

L’IDE pour le projet Python est PyCharm.

## 6.3 - Procédure de packaging et de livraison

---

L’application fera l’objet d’un déploiement sur le serveur OVH au moment de la livraison finale.  
Les identifiants et mots de passe seront donnés au client après la mise en ligne et le paiement de ce dernier.

## 7 - GLOSSAIRE

<b>MPD</b>	Modèle Physique de données
<b>SGDB</b>	Système de Gestion de Base de Données
<b>TCP/IP</b>	La suite TCP ( <i>transmission Control Protocol</i> ) /IP ( <i>Internet Protocol</i> ) est l'ensemble des protocoles utilisés pour le transfert des données sur Internet
<b>SSH</b>	Secure SHell (SSH) est à la fois un programme informatique et un protocole de communication sécurisé.
<b>IDE</b>	Dans un environnement de développement « intégré » (abrégé EDI en français ou IDE en anglais, pour integrated development environment).
<b>PyCharm</b>	Il permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration de logiciel de gestion de versions, et supporte le développement web avec Django.