OC Pizza

IT Consulting & Development|

OC Pizza

SauceTomapp

Dossier de conception technique

Version 1.0.0

Auteur
Louise Zanier-Roumieux
Analyste-programmeuse

Table des matières

1 - Versions

Auteur	Date	Description	Version
Louise Z.	02/05/2020	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 **SauceTomapp**.

L'objectif du document est de présenter l'implémentation technique de l'application, les outils mis en œuvre, ainsi que les technologies utilisées.

Les éléments du présents dossiers découlent :

- De l'entretien avec les fondateurs d'OC Pizza du 01/12/2020
- De l'analyse des besoins déjà effectuée par IT Consulting & Development
- De la rédaction du dossier de conception fonctionnelle.

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 Technique

La pile logicielle est la suivante :

- Application développée en **Python** 3.8, avec le framework **Django** 3.0.6.
- Serveur HTTP **Nginx**, avec **uWSGI**

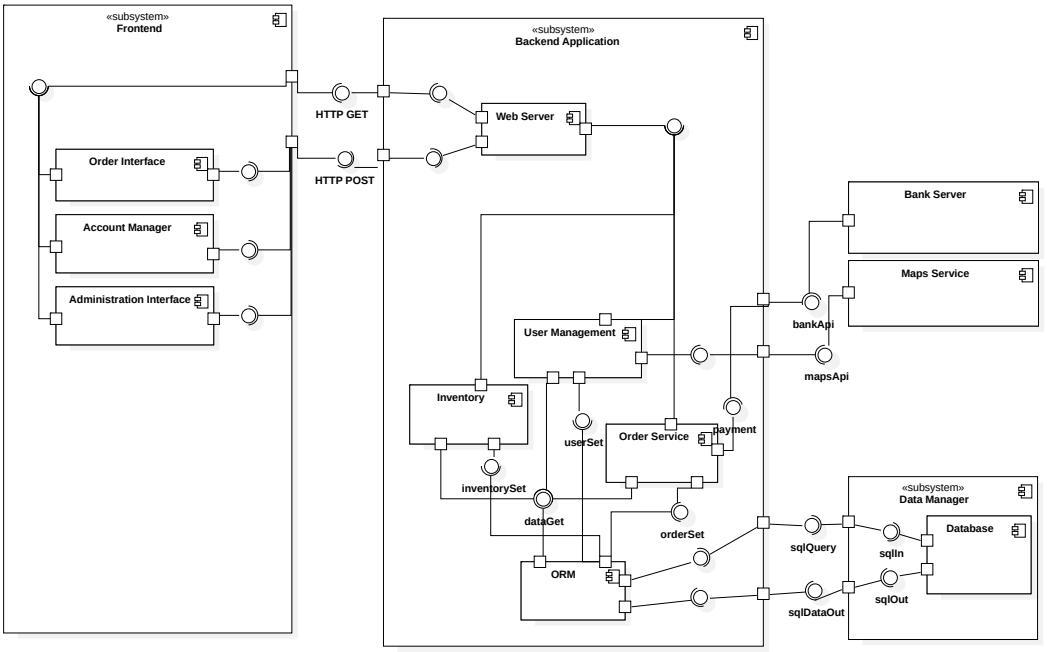

Figure 1: Diagramme de composants de l'application

3.1 - **Composants Order Interface, Account Manager, et Administration Interface**

Ils sont implémentés en HTML, CSS et JavaScript, et permettent à eux trois l'interaction avec le serveur. Les fonctionnalités sont découpés de la même manière que dans le Dossier de conception fonctionelle.

Ils sont affichés sur le navigateur de l'utilisateur, sont servis par le serveur Web et lui envoie les informations données par l'utilisateur.

3.2 - **Composants Web Server**

Le serveur Web utilisé est Nginx. Ce choix a été fait de par sa légèreté, qui lui permet de gérer suffisamment de requêtes sans jamais devoir penser à comment gérer plus, même en ouvrant plusieurs autres pizzerias. Il interagit avec les composants du front-end par le protocole HTTP (par dessus le protocole TCP).

3.3 - **Composant User Management**

Le composant *User Management* est implémenté comme une application du projet Django, et contient toutes les fonctionnalités nécessaires à gérer les utilisateurs. Il permet donc aux utilisateurs de s'inscrire, de se connecter, ou de gérer son compte.

C’est par ce composant que l’interface d’administration passe aussi pour modifier les utilisateurs.

Il interagit avec le *Web Server* par le biais de WSGI, comme les autres composants. Le composant interagit également avec le service de cartographie pour assigner une pizzeria à une adresse.

3.4 - **Composant Order Service**

Le composant *Order Service* est aussi implémenté comme une application du projet Django, et contient cette fois les fonctionnalités nécessaires à passer commande, et pour les employés, à préparer les commandes. C’est là que sont la plupart des fonctionnalités que vont utiliser les employés, à l’exception du gestionnaire de points de vente, pour lequel la seule utilité de ces fonctionnalités va être de connaître les chiffres des commandes.

On retrouve donc la logique derrière la prise de commande, la logique de paiement, ou encore celle de prise en charge par les cuisiniers et les livreurs.

Comme les autres applications du projet Django, il interagit avec le *Web Server* par le biais de WSGI. Il interagit également avec le service bancaire pour gérer les paiements en ligne.

3.5 - **Composant Inventory**

Le composant *Inventory*, encore une fois, est une application du projet Django. Il contient les produits qui existent dans le service, les menus, les recettes des produits, ainsi que l’inventaire des ingrédients de chaque pizzeria.

C’est donc ce composant qui va être interrogé pour savoir quel produit peut être commandé, ou qui va donner la recette si le cuisinier en a besoin.

3.6 - **Composant ORM et Database**

L’*ORM* est le composant qui est connecté aux composants *User Management*, *Order Service* et *Inventory*.

Il permet d’accéder aux données dans la base de données relationnelle, qu’on détaillera dans une partie suivante, en fournissant des interfaces à ces composants pour demander les données de façon idiomatique, et pour recevoir en réponse des objets faciles à manipuler.

Il permet aussi d’avoir une abstraction, et de ne pas devoir réécrire tout les composants si l’on devait changer de système de base de données, ce qui permet une plus grande réusabilité du système complet.

Le composant *Database* est une simple modélisation de la base de données relationnelle, avec laquelle l’*ORM* interagit par le biais de requêtes SQL à travers des sockets.

3.7 - **Composant Bank Service et Maps Service**

Ces composants sont de simples modélisations d’API externes, plus précisément du service bancaire et du service de cartographie. Ils ne font pas partie de l’application à développer, mais elle interagit avec eux afin d’obtenir des informations.

Le service bancaire est utile pour gérer les paiements en ligne, comme savoir si le

paiement est accepté ou refusé.

Le service cartographique sert à associer une pizzeria à une adresse, pour savoir quelle pizzeria va préparer la commande.

4 - Architecture de Déploiement

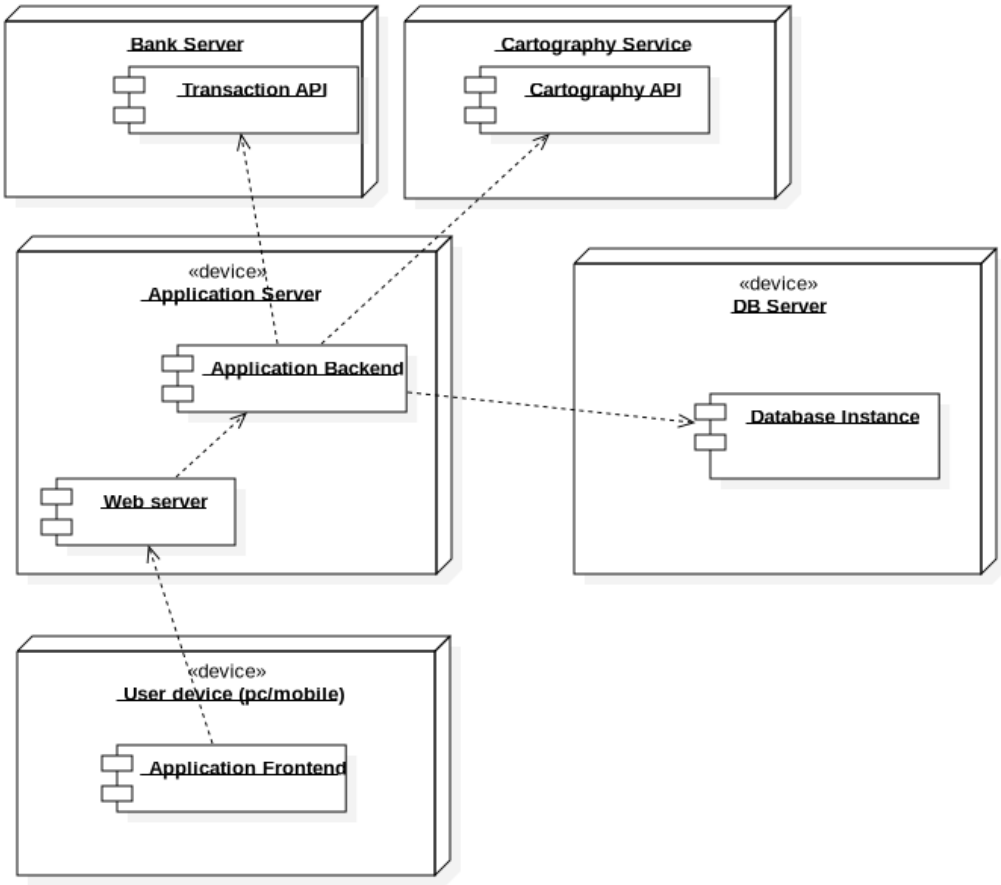

Figure 2: Diagramme de déploiement de l'application

Le diagramme de déploiement est assez transparent compte tenu du diagramme de composants, les plus grands composants sont repris, et les composants *User Management*, *Order Service*, *Inventory* et *ORM* sont rassemblés en un seul, *Application Backend*, qui représente globalement le projet Django.

4.1 - **Serveur de Base de données**

Le serveur de base de données utilisé sera dans un conteneur **Docker**, qui peut donc soit tourner sur la même machine que l'application, soit sur une machine différente, mais qui sera dans tout les compartementalisé. Le serveur SGDB utilisé sera *PostgreSQL*, un système de gestion de base de données relationnelle, permettant donc, vu le diagramme de classes abordés dans le dossier de conception fonctionnelle, de gérer parfaitement ce dont l'application a besoin.

PostgreSQL a été choisi pour ses fonctionnalités permettant de potentiellement agrandir l'application, comme par exemple la réplication entre plusieurs serveurs, ou encore de meilleures performances, qui ne seront pas forcément notables dans l'état actuel, mais qui seront intéressantes quand le client, comme abordé dans l'entretien, voudra agrandir son réseau de pizzerias.

5 - Architecture logicielle

5.1 - **Principes généraux**

On utilise plusieurs outils pour gérer le code :

- Le versionnage du code est géré par **Git**.
- Les dépendances Python sont gérées avec **Pip**, le gestionnaire de paquets Python, qui va permettre d'avoir une réplicabilité du comportement, puisque les versions de chaque dépendances sont gelées.
- Pour les tests unitaires et les tests d'intégration, on utilise **Pytest** et **Coverage**.
- On utilise **Docker** pour gérer l'environnement d'exécution, qu'il soit toujours identique peu importe le déploiement, afin d'éviter au maximum les comportements différents.
- Le déploiement en soit sera géré par **Fabric**.

*5.1.1 - **Les applications***

Un projet Django est divisé en plusieurs applications. Ici, on divisera le projet en suivant les composants définis dans la Figure 1: Diagramme de composants de l'application. Les applications seront donc *User Management*, abbréviée en *users*, *Order Service*, abbréviée en *orders*, et *Inventory*, dont on reprendra le nom.

*5.1.2 - **Les couches***

L'application reprend le modèle Django, le pattern MVT : Modèle-Vue-Template.

- Le **Modèle** est la couche la plus basse du modèle, et le plus réutilisable. Il définit d'abord les données telle qu'elles vont être stockés dans la base de données, mais aussi leurs comportements associés.
- La **Vue**, dans le pattern MVT, est une couche très fine, qui fait le lien entre le **Modèle** et le **Template**. Elle a un minimum de logique associée, se contentant dans le meilleur des cas à uniquement faire un rendu du template, après avoir lancé des comportements du **Modèle**.
- Le **Template** est la couche la moins réutilisable et la plus spécifique à l'application. Elle définit le code HTML qui sera envoyé à l'utilisateur, et comment les données du **Modèle**, parfois (bien que cela doive être réduit au minimum) modifiées par la **Vue**.

*5.1.3 - **Structure des sources***

La structuration des répertoires du projet suit la logique suivante :

– les répertoires sources sont créés de façon à respecter la philosophie Maven (à savoir : « convention plutôt que configuration »)

```
racine
 ├─ Dockerfile
 ├─ docker-compose.yml
 ├─ fabfile.py
```

```
├─ .env
├─ requirements.txt
├─ manage.py
├─ saucetomapp
│   ├─ __init__.py
│   ├─ asgi.py
│   ├─ wsgi.py
│   ├─ urls.py
│   └─ settings
│       ├─ __init__.py
│       ├─ base.py
│       ├─ dev.py
│       └─ production.py
├─ users
│   ├─ __init__.py
│   ├─ admin.py
│   ├─ apps.py
│   ├─ urls.py
│   ├─ views.py
│   ├─ models.py
│   ├─ templates
│   │   └─ users
│   └─ static
│       └─ users
├─ orders
│   ├─ __init__.py
│   ├─ admin.py
│   ├─ apps.py
│   ├─ urls.py
│   ├─ views.py
│   ├─ models.py
│   ├─ templates
│   │   └─ orders
│   └─ static
│       └─ orders
├─ inventory
│   ├─ __init__.py
│   ├─ admin.py
│   ├─ apps.py
│   ├─ urls.py
│   ├─ views.py
│   ├─ models.py
│   ├─ templates
│   │   └─ inventory
│   └─ static
│       └─ inventory
├─ templates
│   └─ base.html
└─ static
    ├─ style.css
    └─ app.js
```

6 - Points particuliers

6.1 - Gestion des logs

Les applications du projet loggent, avec les utilitaires de Django, toutes les erreurs qui arrivent, et qui ne sont pas dûs à l'utilisateur (comme par exemple un formulaire mal rempli, ce qui ça est géré par l'application). En production, les logs sont envoyés par mail à l'administrateur.

6.2 - Fichiers de configuration

6.2.1 - Application web

Toutes les configurations à changer sont gérées par des variables d'environnement, automatiquement set par Docker quand le conteneur est lancé.

6.3 - Ressources

Les ressources graphiques sont fournies par la société cliente, **OC Pizza**. Pour les données de base de l'application, il sera créé un utilisateur pour le client, qui pourra à partir de là créer les autres comptes de ses employés.

6.4 - Environnement de développement

Le scope de ce dévelopement ne justifie pas d'outil particulier outre ceux qui ont déjà été détaillés dans Architecture logicielle.

6.5 - Procédure de packaging / livraison

L'utilisation de **Docker** et de **Fabric** facilitent le packaging et le déploiement. Le Dossier d'exploitation détaille plus amplement leur utilisation dans le cadre de ce projet.

7 - Glossaire

WSGI	Web Server Gateway Interface, une convention d'appel pour permettre l'interaction entre les serveurs webs et les programmes Python.
Socket	Une sortie d'une communication, soit entre processus du même ordinateur (spécifiquement sur Unix), soit entre deux ordinateurs par le biais d'un réseau.