OC Pizza

Projet_6

Dossier de conception technique

Version A

Auteur
Bassem BEN BELGACEM
Ingénieur développement

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
Bassem BEN BELGACEM	30/07/2020	Création du document	A

2 - INTRODUCTION

2.1 - Objet du document

Le présent document constitue le dossier de conception technique de l'application « Système informatique de gestion OC Pizza »

L'objectif du document est de présenter l'architecture technique de l'application en s'adressant aux développeurs et chef projet.

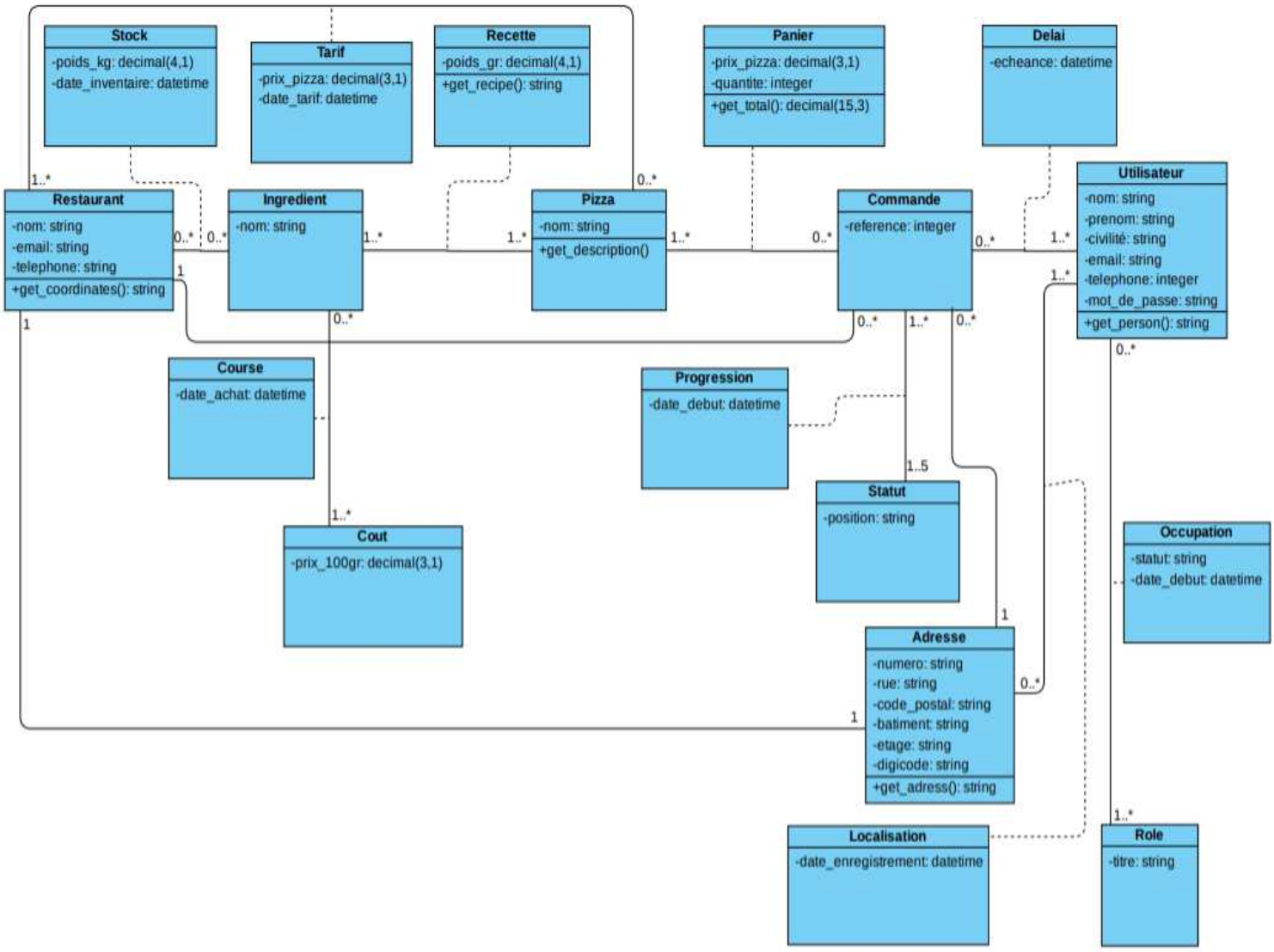
Les éléments du présent dossier découlent de :

- Ppizza1_01_spécfontionnelles

3 - LE DOMAINE FONCTIONNEL

3.1 - Référentiel

Diagramme UML de classes



On présente ici une description du domaine fonctionnel à travers le diagramme de classes.

Comme son nom l'indique, ce diagramme présente les différentes classes et les relations entre elles, dans le but de concevoir la base de donnée de notre application.

Lorsqu'une relation est explicitée dans la description d'une classe A vers une classe B, on ne revient pas à cette relation en décrivant la classe B.

3.1.1 - Règles de gestion

Restaurant : représente les restaurants de OC Pizza
Un restaurant a comme attributs un nom, un email, un numéro de téléphone ainsi qu'une adresse.
La relation entre la classe restaurant et la classe Adresse et one-to-one, car un restaurant a une seule adresse et une adresse correspond à un seul restaurant. On a choisi de séparer Adresse comme une classe à part car ce serait une entité nécessaire à d'autres classes avec des relations différentes.

Un restaurant contiendra dans sa cuisine plusieurs ingrédients et un ingrédient peut exister dans plusieurs restaurants. D'où la relation many-to-many entre la classe restaurant et **Ingrédient**. Entre les deux la classe d'association **Stock** contient le poids en kg de chaque ingrédient dans un tel restaurant ainsi que la date de l'inventaire.

Un restaurant peut être à 0 ou plusieurs commandes mais une commande ne peut sortir que d'un seul restaurant. D'où la relation one-to-many entre Restaurant et **Commande**.

Un restaurant peut livrer plusieurs pizzas et une pizza sort au moins d'un restaurant. D'où la relation many-to-many entre les classes Restaurant et **Pizza.**
Entre les deux classes, la classe d'association **Tarif** comporte le prix d'une telle pizza dans un tel restaurant ainsi que la date de ce tarif**.**

Ingrédient : représente les ingrédients qui composent les pizzas chez OC Pizza
Un ingrédient aura comme attribut un nom.
Un ingrédient peut avoir des coûts variés suivant les jours dans un marché et un seul coût peut s'affecter à plusieurs ingrédients. D'où la relation many-to-many entre la classe Ingrédient et la classe **Cout.**
Entre ces deux classes, la classe d'association **Course** comporte la date d'achat comme attribut.

Un ingrédient peut être présent dans au moins une pizza et une pizza contient au moins un ingrédient.
D'où la relation many-to-many entre la classe Ingrédient et la classe **Pizza**.
Entre ces deux classes, la classe d'association **Recette** comporte le poids en gr d'un tel ingrédient dans une telle pizza.

Pizza : représente les pizzas vendues chez OC Pizza
Une pizza aura comme attribut un nom.
Une pizza peut s'associer à plusieurs commandes et une commande aura au moins une pizza.
D'où la relation many-to-many entre la classe Pizza et la classe **Commande**.
Entre les deux, la classe d'association **Panier**, comporte le prix d'une telle pizza au sein d'une telle commande ainsi que la quantité commandée d'une même pizza.

Commande : représente les commandes délivrées par OC Pizza
Une commande aura comme attribut une référence.
Une commande s'associe à au moins un utilisateur que ce soit client ou staff OC Pizza, et un utilisateur peut gérer plusieurs commandes. D'où la relation many-to-many entre les classes Commande et **Utilisateur**.
Entre les deux, la classe d'association **Délai**, porte l'échéance que le client précise pour avoir son pizza.

Une commande peut avoir plusieurs statuts qui sont les différents états depuis l'initiation de la commande par un client jusqu'à la livraison. Un état peut concerner plusieurs commandes. On conclut une relation many-to-many. Entre les deux classes Commande et **Statut**, la classe d'association **Progression** trace la date de début de chaque état.

Une commande est forcément liée à une seule adresse et une adresse peut concerner plusieurs commandes. On conclut une relation one-to-many entre les classes **Adresse** et Commande.

Utilisateur : représente les utilisateurs de OC Pizza à savoir les clients ou le personnel
Un utilisateur est qualifié par un nom, un prénom, une civilité, un email, un numéro de téléphone et un mot de passe.
Un utilisateur peut avoir une ou plusieurs adresses et une adresse peut concerner plusieurs utilisateurs.
D'où la relation many-to-many entre les classes Utilisateur et **Adresse**.
Entre ces deux classes, la classe d'association **Localisation** trace la date d'enregistrement d'un tel utilisateur à une telle adresse.

Un utilisateur aura au moins un rôle et un rôle peut être associer à plusieurs utilisateurs. D'où la relation many-to-many entre les classes Utilisateur et **Rôle.**
Entre ces deux classes, la classe d'association **Occupation** précise si l'utilisateur est actif ou pas, par rapport à son rôle ainsi que la date de début de ce statut.

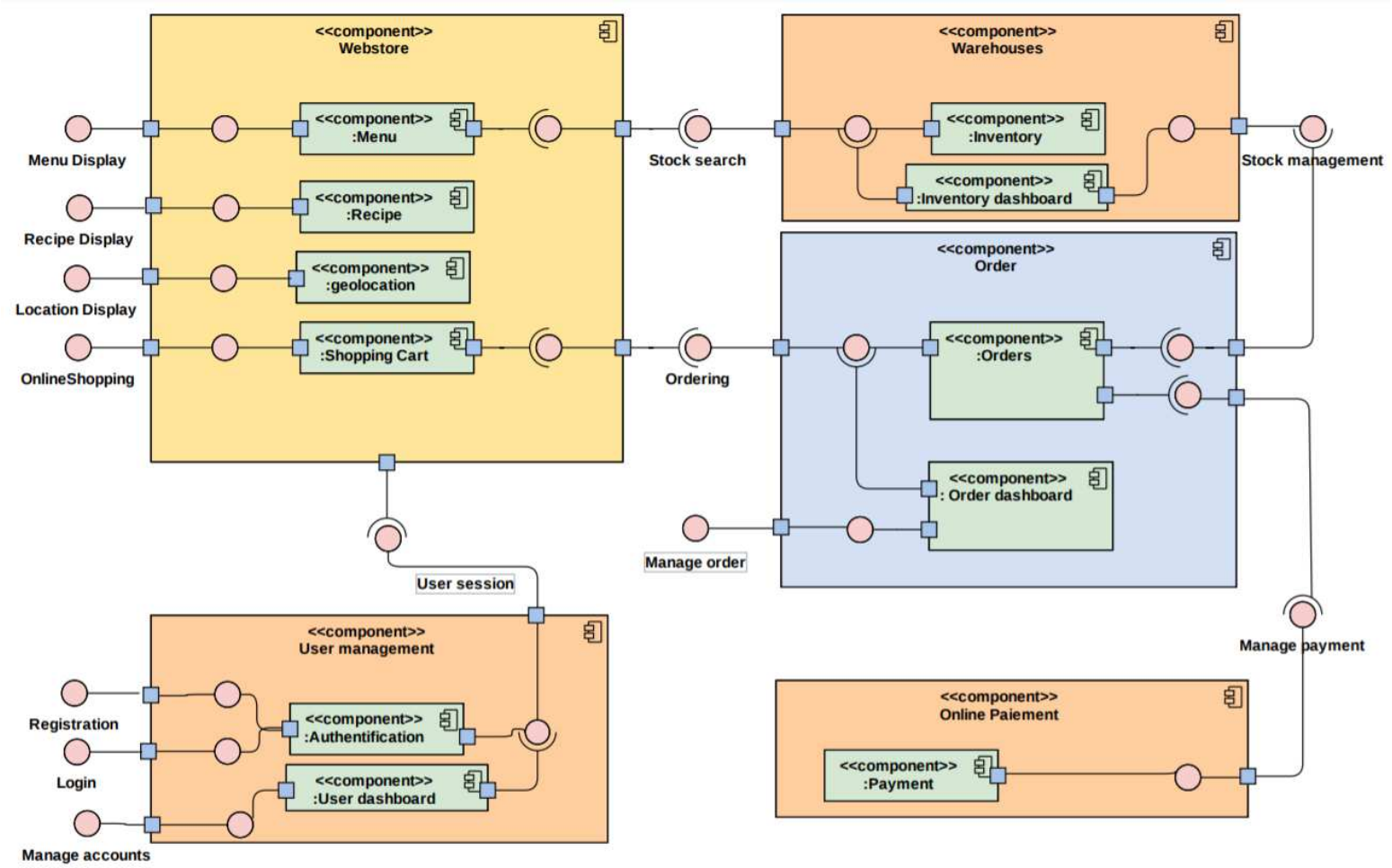
4 - ARCHITECTURE TECHNIQUE

4.1 - Application Web

La pile logicielle est la suivante :

- Application **Python 3.8 et sa librairie Django 3 pour le back end, HTML 5/CSS/Bootstrap et Javascript 11 et sa librairie jQuery 3.5.1, pour la partie front-end.**
- Serveur d'application **Gunicorn.20.0.4**
- Serveur web **NGINX 1.18.0**

Diagramme UML de Composants



4.1.1 - Subsystem Webstore

La structure interne de « Webstore » comporte quatre composants :

- Le composant Menu qui fournit la carte ou menu et uilise comme input nécessaire à son fonctionnement, l'état du stock pour afficher seulement ce qui est susceptible d'être préparé suivant les ingrédients disponible.
- Le composant « Recipe » ou recette expose les recette pour chaque pizza
- Le composant « geolocation » fournit une position du restaurant OC pizza le plus proche suivant l'adresse fournie par un client
- Le composant « Shopping cart » ou panier fournit l'interface de commande en ligne et utilise le résultat du composant « Orders »

4.1.2 - Subsystem Warehouse

La structure interne de cette sous famille comporte:

- Un composant « Inventory » ou inventaire pour gérer et calculer l'état du stock. Ce composant fournit les informations necessaires pour le composant menu afin de montrer les produits disponibles.
- Un composant « Inventory dashboard » pour le suivi de l'état du stock. Ce composant a besoin de l'information qui découle du composant de la même sous famille à savoir « Inventory » et fournit la gestion de stock indispensable pour valider les commandes

4.1.3 - Subsystem Order

La structure interne de cette sous famille comporte:

- Un composant « Orders » pour gérer les commandes et fournir les informations nécessaires au composant « Order dashboard ». Le composant « Order » a besoin des informations de paiement et de l'évolution de l'état du stock.
- Un composant « Order dashboard » pour le suivi de l'état des commandes. Ce composant a besoin de l'information qui découle du composant de la même sous famille à savoir « Orders » et fournit un apperçu pour suivre les commandes en temps réel.

4.1.4 - User management

Cette sous famille concerne l'authentification et la gestion des comptes et comporte deux composants :

- Un composant « Authentification » qui fournit deux output, une pour la connexion d'un utilisateur et l'autre nécessaire à la sous famille « webstore » pour se positionner sur un compte et avoir des accès relatifs à cette session.
- Un composant « User dashboard » qui utilise les données fournies par le composant « Athentification » et fournit un tableau de bord pour gérer les collaborateurs et les clients

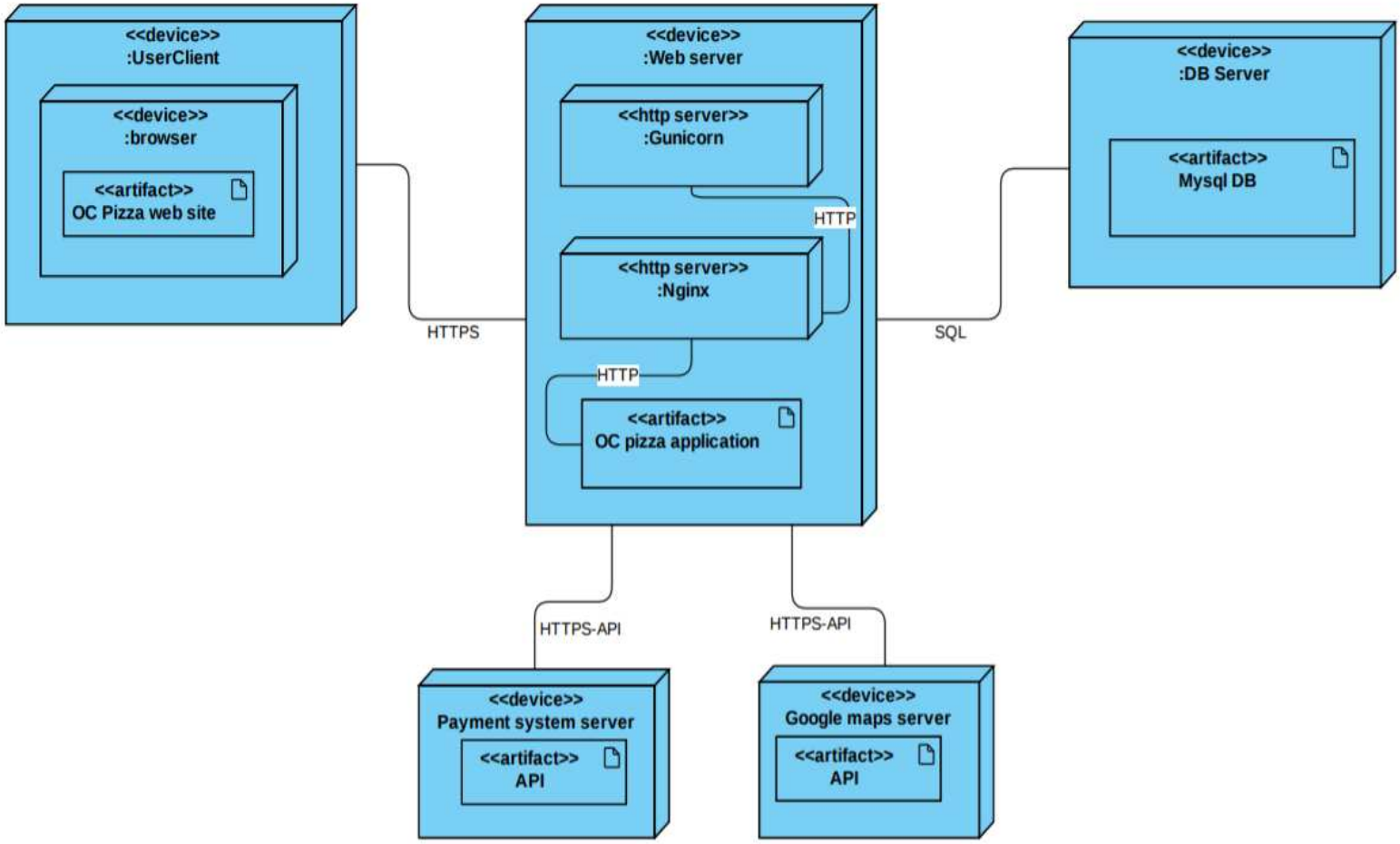
4.1.5 - Online paiement

La structure interne comporte un seul composant qui est « Payment ».

Ce composant fournit une information indispensable au fonctionnement du composant « Orders »

5 - ARCHITECTURE DE DÉPLOIEMENT

Diagramme UML de déploiement



Le diagramme de déploiement sert à présenter l'architecture physique de notre application, en montrant quels éléments logiciels sont déployés sur quels éléments matériels

5.1 - Appareil client

L'utilisateur pourra communiquer avec l'application du restaurant par le biais de son ordinateur et doit être connecté à internet et possède un navigateur web pour que ce dernier puisse faire les requêtes au serveur de l'application.

Les requêtes se feront suivant un schéma *request-response* en respectant le protocole *https*.

Le browser convertit les pages front end fournit par le serveur de l'application en un aperçu compréhensible par l'utilisateur.

5.2 - Serveur web

Le serveur web hébergera:

- Un serveur web http *Nginx* qui reçoit les requêtes *https* et les transfère au serveur d'application *Gunicorn.* Il fait aussi parvenir les réponses http de l'application du restaurant au navigateur web du client
- Le serveur d'application http *Gunicorn* qui travaille en association avec *Nginx*
- L'application *OC pizza* qui fonctionne avec *python 3.8* et *Django 3*. *Django* reçoit les requêtes de *Nginx* et lui renverra les réponses http.

5.3 - Serveur de Base de données

On utilise un serveur de base de donnée relationnelle qui est Mysql.

Ce serveur répond parfaitement à notre besoin, maitrisé par les membres de l'équipe et il est gratuit.

L'artéfact, dans ce cas, est la base de donnée conçue en fonction de la spécification fonctionnelle de ce projet.

Le serveur web et la base de donnée communiquent via le language *sql.*

5.4 - Serveur externes

Le serveur du système de paiement (banques ou *paypal*) communique avec le serveur web suivant le protocole *https* en s'adressant à l'API de l'organisme de paiement en question.

De même pour le serveur de *google maps*.

6 - GLOSSAIRE

API	Application programming interface