OC Pizza

Projet 6

Dossier de conception technique

Version 1.1

Jonathan REVEILLE
Développeur Application Python Junior

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
J.REVEILLE	20/02/2020	Création du document	0.1
J.REVEILLE	21/02/2020	Ajout diagramme de classe et notes	0.2
J.REVEILLE	22/02/2020	Ajout diagramme de composants et notes	0.3
J.REVEILLE	23/02/2020	Ajout notes sur les serveurs (web, application, base de données)	0.4
J.REVEILLE	23/02/2020	Ajout diagramme de déploiement et notes	0.5
J.REVEILLE	24/02/2020	Nouveau diagramme de déploiement (ajustement et nœuds)	0.6
J.REVEILLE	24/02/2020	Dossier prêt – attente validation mentor	0.9
J.REVEILLE	25/02/2020	Validé par mentor – attente échange avec développeur interne	1
J.REVEILLE	27/02/2020	Ajustement et validation par mentor	1.1

2 - INTRODUCTION

2.1 - Objet du document

Le présent document constitue le dossier de conception technique de l'application OC Pizza. Suite au dossier de conception fonctionnel (cf. dossier de conception fonctionnel version 1.1 Projet 4), il est important de mettre en œuvre et de décrire la représentation des différents composants qui feront l'application d'OC Pizza. Cela nous permettra donc de modéliser les objets du domaine fonctionnel et d'identifier les différents éléments composant le système à mettre en place et leurs interactions. Ce document présentera la description du déploiement des différents composants que nous envisageons.

L'objectif du document est de décrire les aspects techniques de l'application et de fournir des explications concernant les futurs composants de celle-ci afin de comprendre leur comportement, et les différents moyens de communications qui existeront.

Nous allons utiliser plusieurs diagrammes de la méthodologie UML afin de schématiser au plus proche l'aspect technique de la future application, qui elle, gèrera le système de gestion de pizzeria chez OC Pizza. Nous verrons dans ce dossier plus précisément le **diagramme de classe, le diagramme de composant et le diagramme de déploiement** qui nous donneront une vision du plus large au plus précis sur le fonctionnement de l'application future. Afin d'être au plus proche du futur système, nous avons également élaboré une **base de données** temporaire prenant en compte toutes les idées de développement sur l'aspect technique de la solution (présent dans ce dossier livré), et elle nous permettra aussi de se projeter sur la future base de données de notre application afin la visualiser et de tester son bon fonctionnement. Un modèle physique des données est présent dans le dossier du livrable.

Dans un premier temps, le **diagramme de classe UML** nous permettra d'exposer les différentes relations existantes entre les classes que composera l'application en fonction des besoins exprimés par le client (cf. diagramme de classe). Cette étape nous permet de comprendre les relations existantes entre les différentes classes sur lesquelles nous nous baserons pour construire notre base de données pour l'application web.

Dans un second temps, nous verrons également dans ce dossier comment chacun de ses composants existeront dans l'application et de comprendre comment ils agiront entre eux (leurs interactions). Par ici, nous entendons les interfaces proposées et les interfaces requises par composant (cf. **diagramme de composant UML**). Cela aidera à obtenir une meilleure compréhension de la responsabilité précise de chacun des composants (ce qu'un composant propose et ce qu'un composant requiert pour fonctionner, équivalent à leurs interactions).

Enfin, dans un dernier temps, nous irons voir notre **diagramme de déploiement** qui nous permet de prendre du recul sur l'application et de décrire le fonctionnement qui existera entre les différentes machines (périphérique, serveur…) qui se connectent à notre application web. Ce diagramme nous permettra de mettre en œuvre les différents serveurs, les différents environnements d'exécution et les différents périphériques qui seront concernés. Nous verrons plus en détail dans ce dossier les différents choix de technologies implémentés pour notre future application web.

Les éléments du présent dossier découlent : du document écrit et des discussions avec le client, ainsi que le « Recueil des besoins » par le client et le dossier de conception fonctionnel du projet 4 d'Openclassroom (v.1.1).

3 - LE DOMAINE FONCTIONNEL

3.1 - Référentiel

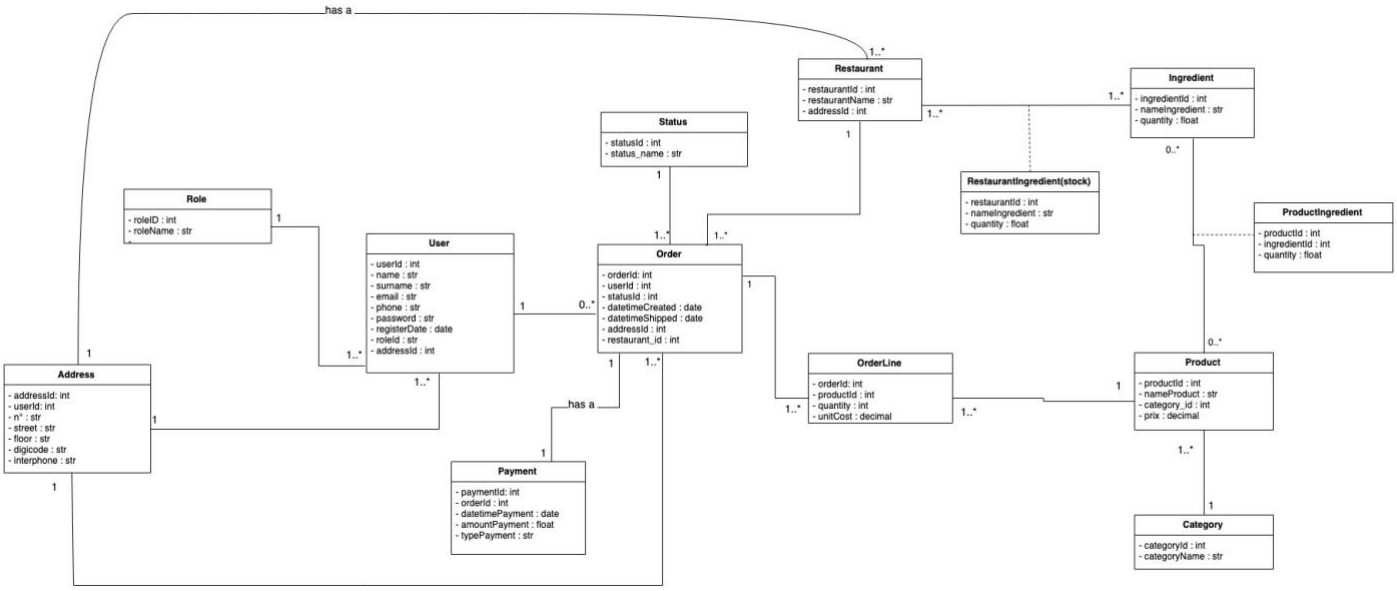


Diagramme de classe UML

Notre application est composée de plusieurs classes pour son bon fonctionnement. Nous pouvons déjà compter 11 classes et 2 classes qui feront office d'association. Ces différentes classes nous permettront d'archiver des données (en masse) afin que notre application puisse fonctionner, évoluer et de faciliter sa maintenance. Nous détaillerons dans la partie suivante les relations entre les différentes classes qui composera notre application et qui reflètera les relations entre les classes dans notre base de données.

3.1.1 - Règles de gestion

- Pour la classe « **Order** »

Une commande aura un paiement et un paiement concernera (pointera vers) une commande uniquement. Cela explique la *relation un à un* entre la classe **« Order »** et la classe **« Payment ».** Une commande sera toujours rattachée à une adresse **(« Address »)** uniquement, tandis qu'une adresse peut apparaître pour plusieurs commandes (*relation un à plusieurs).* Une commande pointera vers un utilisateur **(« User »)**, et un utilisateur peut faire plusieurs commandes (*relation un à plusieurs*). Une commande aura un status **(« status »)**, alors que les status pourront être appliqué à

plusieurs commandes (*relation un à plusieurs*). Une commande pourra être attribué à un restaurant **(« Restaurant »)** tandis qu'un restaurant pourra recevoir plusieurs commandes (*relation un à plusieurs*). Enfin, une ligne de commande **(« Orderline »)** pointera vers une commande, et une commande sera représenté par une ou plusieurs lignes de commandes (*relation un à plusieurs).*

- Pour la classe **« OrderLine »**

Une ligne de commande pourra contenir une pizza ou un type de produit **(« Product »)**. Une pizza (ou un type de produit) pourra être présente dans une à plusieurs lignes de commande (*relation un à plusieurs*).

- Pour la classe **« Product »**

Un produit, ou une pizza, pourra appartenir à une seule catégorie de produit **(« Category »).** Une catégorie pourra ressourcer un à plusieurs produits (*relation un à plusieurs).* En ce qui concerne la classe **« Ingredient »,** nous avons donc ici avec la classe **« Product »** une *relation plusieurs à plusieurs.* Cela nécessite de créer une classe d'association entre les deux, avec des clés étrangères qui pointent vers les instances de chacune des classes.

- Pour la classe **« Restaurant »**

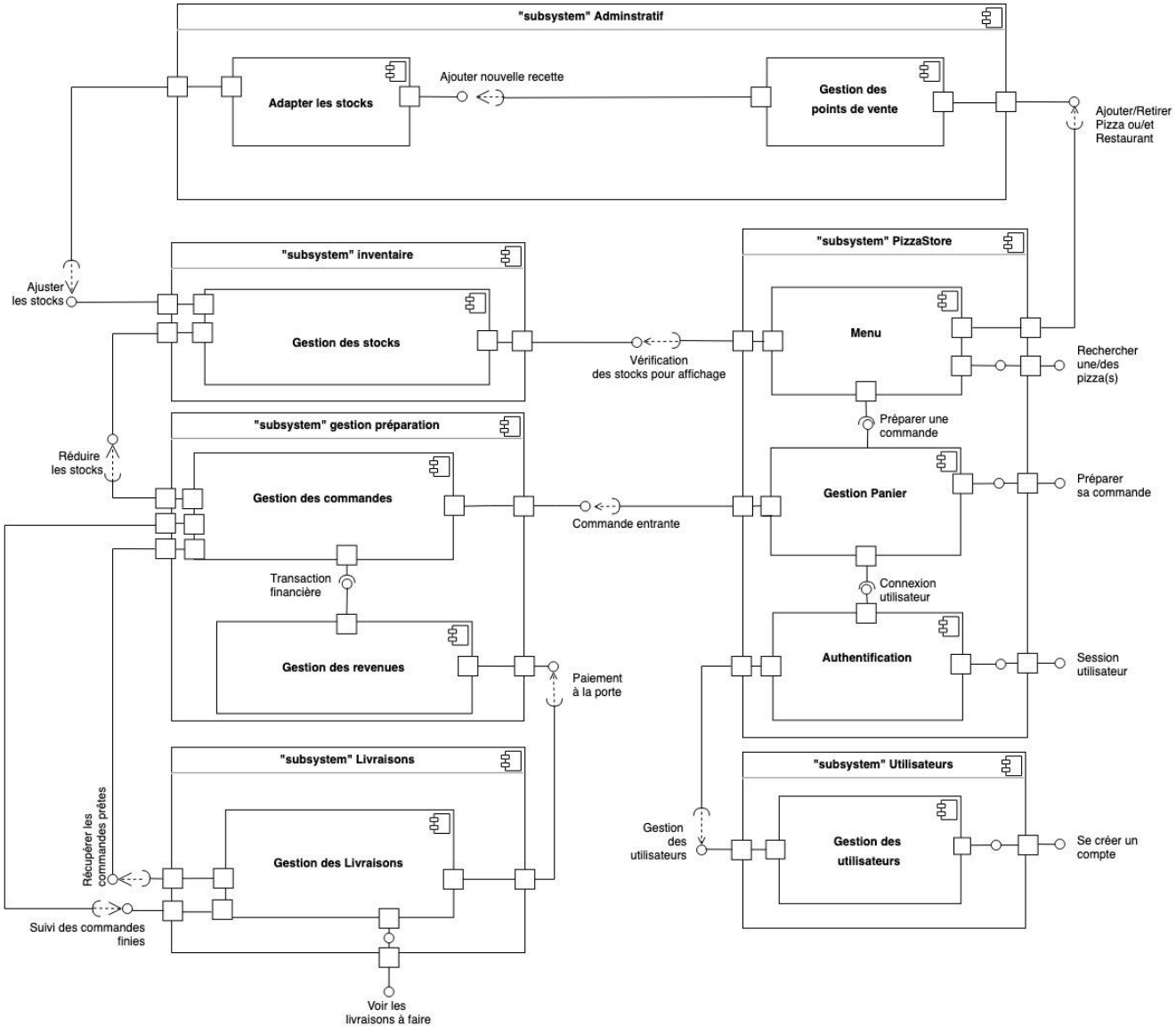
Un restaurant aura plusieurs ingrédient **(« Ingredient »),** et les ingrédients seront dans plusieurs restaurants, ce qui explique une autre *relation plusieurs à plusieurs.* Une classe d'association émerge avec les clés étrangères de chacune de ses deux classes, ainsi que les quantités d'ingrédients y figure.

- Pour la classe **« User »**

Un utilisateur aura un rôle (client, employé, livreur, ou admin), et un rôle **(« Role »)** aura plusieurs utilisateurs, nous avons donc une *relation un à plusieurs.* Pareillement pour les adresses, un utilisateur aura une adresse **(« Address »)** et les adresses pourront contenir plusieurs utilisateurs, ceci explique la relation *un à plusieurs*.

4 - ARCHITECTURE TECHNIQUE

4.1 - Application Web



Tous les composants dans ce diagramme sont connectés à l'ORM (Peewee ou SQLAlchemy)

Diagramme UML de Composants

Les composants représentent des parties « logiciel » de notre future application. Nous verrons dans cette partie du dossier que chaque composant à une ou plusieurs responsabilités qu'on déterminera avec le terme d'interface (proposer et requérir). Il faut savoir que chaque composant est une partie de notre programme qui est indépendante du reste.

Dans la réalisation de ce diagramme, nous découpons notre logiciel par entité, c'est-à-dire une entité qui gèrera toute l'interface graphique de prise de commande, une autre entité qui gèrera les commandes, une autre entité qui gèrera les livraisons, une autre qui gèrera l'inventaire, une autre pour les utilisateurs et enfin une qui gèrera l'administration. On peut comprendre qu'à partir de cette description qu'un composant peut proposer ou peut faire appel à des méthodes (fonctions) venant des autres composants. Pour vulgariser, nous pouvons voir un composant comme une classe qui comporte des méthodes (fonctions) et qui peut faire appel à d'autres méthodes provenant des autres classes (composant). Afin de comprendre le diagramme, le signe du composant qui propose une interface (propose une méthode) est dessiné par un rond. Le composant qui fait appel à une méthode est dessiné par un demi-cercle. C'est de cette manière que les composants ont des interactions entre eux.

Nous allons décrire dans la prochaine partie chacun des composants : leur rôle, leur objectif et ainsi que la description des interfaces qu'elle propose et requiert.

4.1.1 - Les composants par entité

Subsystem (entité) : *PizzaStore*

- **Composant Menu** : Ce composant menu permet à la console de l'utilisateur de faire une recherche des pizzas disponibles pour le restaurant sélectionné. Ce composant permet également de faire appel (requiert) à des méthodes des autres composant. Le composant menu fait appel (requiert) en premier vers l'entité qui gère l'inventaire (les stocks) afin de déterminer quelles sont les pizzas étant encore disponibles par restaurant, afin que l'application affiche bien les pizzas réalisables pour la commande de l'utilisateur. Enfin, ce composant peut aussi faire appel (requiert) à l'entité Administratif afin de pouvoir ajouter ou retirer des pizzas selon les saisons, ou pendant d'éventuelles pénuries de stocks. Ce composant pourra aussi faire appel (requiert) à la Gestion du panier, afin que l'utilisateur puisse se constituer un panier pour une potentielle future commande sur l'application.
- **Composant Gestion Panier** : Ce composant permet à l'application et à l'utilisateur de préparer sa commande. Il propose comme interface de préparer une commande (propose), il aidera donc le composant Menu à créer un panier afin qu'on puisse procéder à une commande plus tard. Ce composant fera appel (requiert) au gestionnaire des commandes afin de pouvoir envoyer sa commande lorsqu'elle sera complète. Il fera aussi appel (requiert) à la méthode pour s'identifier auprès du composant Authenfication avant que l'utilisateur puisse envoyer sa commande au restaurant sélectionné.
- **Composant Authenfication** : ce composant proposera comme interface une connexion pour les utilisateurs, que ce soit avant la prise de commande ou pendant la réalisation d'un panier de commande. Il fera appel (requiert) au composant de la gestion des utilisateurs, pour valider la connexion, ou bien, si un utilisateur est nouveau et souhaite se créer un compte sur notre application web d'OC Pizza.

Subsystem (entité) : *Utilisateur*

- **Composant Gestion des utilisateurs :** Ce composant permet aux utilisateurs de se créer un compte à partir de leur console s'il sollicite directement cette partie logicielle de notre application. La gestion utilisateur proposera en interface la gestion des utilisateurs afin de vérifier si l'utilisateur existe, ou si, il nécessite de se créer un compte sur l'application.

Subsystem (entité) : *Livraison*

- **Composant Gestion des livraisons** : ce composant permet au livreur de pouvoir visualiser quelles sont les livraisons à réaliser. Ce composant fait appel (requiert) à la méthode du composant Gestion des revenues afin de faire payer à la livraison la commande. Il requiert également une interface qui fait appel au composant de Gestion de commande afin de pouvoir récupérer les commandes prêtes à partir du restaurant, afin qu'elles passent au status de livraison en cours. La gestion des livraisons propose comme méthode le suivi des livraisons au composant de gestion des commandes. Cela permettra au gestionnaire de commande de savoir exactement où en est l'acheminement d'une commande une fois qu'elle a quitté le restaurant.

Subsystem (entité) : *Gestion des préparations*

- **Composant des commandes** : ce composant propose en méthode au composant des livraisons de récupérer les commandes entrantes. Il va à son tour demander au composant des Livraisons quelles sont les livraisons finalisées. Le composant des commandes fait appel (requiert) au composant des stocks afin de réduire les stocks d'ingrédients utilisés lorsqu'une commande est entrante et qui a été validé par le système.
- **Composant des revenues :** ce composant est doté de deux interfaces qui propose aux composants Livraison et Commandes de pouvoir procéder au paiement pour une commande.

Subsystem (entité) : *Inventaire*

- **Composant gestion des stocks** : ce composant détient trois méthodes qu'il propose aux composants Menu, Adapter les stocks, Gestion des commandes. La première étant de fournir (propose) au menu les stocks disponibles afin que l'affichage des pizzas disponibles soit en fonction des stocks par restaurant. La deuxième méthode qu'il propose est d'ajuster des stocks pour le composant Adapter les stocks (réapprovisionnement ou ajout d'un nouvel ingrédient). Et enfin la dernière méthode (propose) est de réduire les stocks lorsqu'une commande est entrante.

Subsystem (entité) : *Administratif*

- **Composant Adapter les stocks :** ce composant propose d'ajouter de nouvelle recette dans la liste des stocks (ensemble des ingrédients nécessaires pour un produit transformé), et

enfin il fait appel (requiert) aux stocks pour pouvoir ajuster les stocks (si ajout d'une nouvelle matière première et sa quantité par exemple).

- **Composant Gestion des points de vente :** ce composant requiert une interface pour ajouter ou retirer des recettes (produit) par exemple pour un ou plusieurs points de ventes. Ce composant propose l'interface de pouvoir ajouter ou retirer des pizzas, ou bien, des points de vente, au composant Menu se situant dans l'entité PizzaStore.

5 - ARCHITECTURE DE DEPLOIEMENT

La pile logicielle est la suivante :

- Application : **Python 3.7**
- Framework : **Django 3.0.3**
- Serveur d'application : **Gunicorn 20.0.4**
- Serveur web : **Nginx 1.17.0**
- Serveur base de données : **MySQL 8.0**
- OS **Ubuntu 18.04**

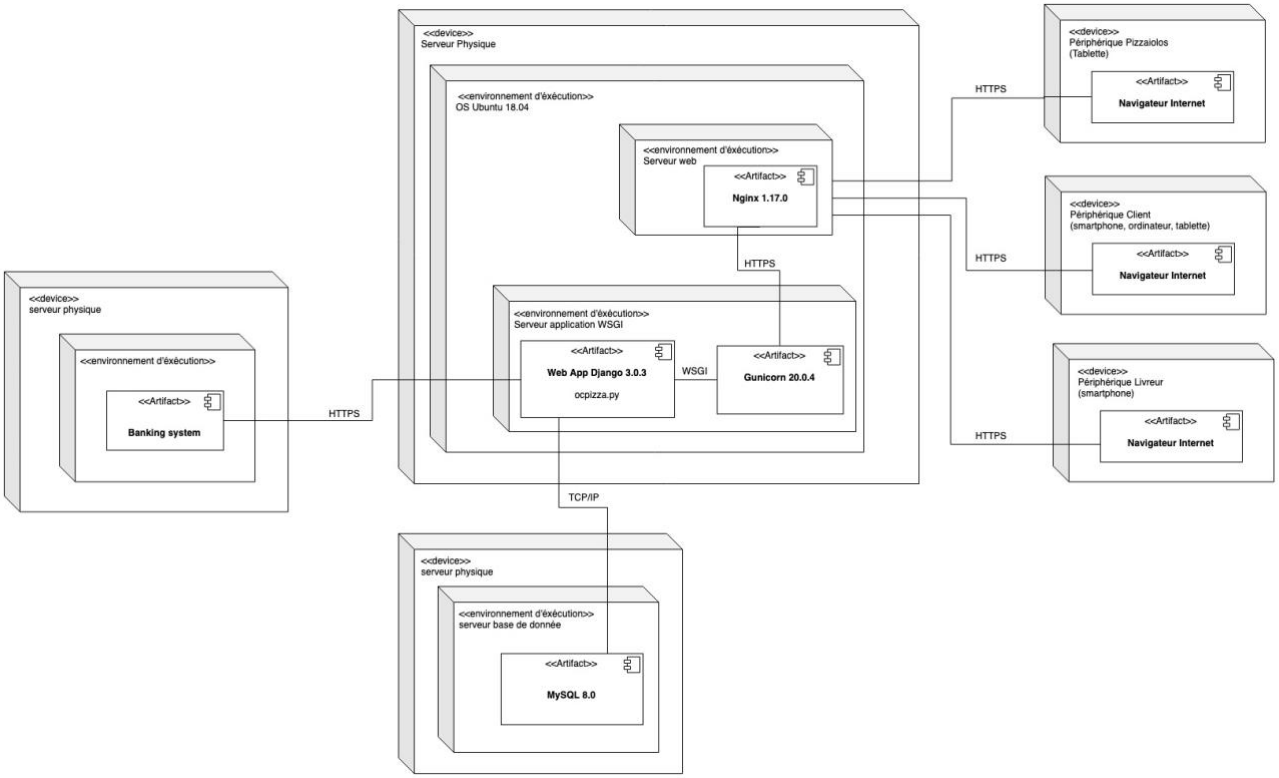


Diagramme UML de déploiement

Le diagramme de déploiement représente les serveurs, les périphériques et les connexions entre eux pour notre application.

Nous pouvons voir que pour se connecter à notre application, il nous faudra uniquement le nom de domaine à partir de n'importe quel périphérique et une connexion internet. Nous avons la même porte d'entrée pour tout type d'utilisateurs. Selon les différents profils d'utilisateurs qui accèderont

à l'application web, ils auront une interface personnalisée. Tous accèderont par différents points d'entrées à l'application, c'est-à-dire, qu'il existera plusieurs URL pour accéder à notre application OC Pizza selon le profil d'utilisateur. Chacun aura une différente interface étudiée afin de faciliter son utilisation. Les protocoles de connexion se feront en majeure partie en HTTPS entre les différents nœuds et seulement le protocole de connexion à la base de données MySQL s'effectuera en TCP/IP.

5.1 - Serveur de Base de données

MySQL 8.0 : Pour la réalisation de notre application, nous avons sélectionné comme choix de base de données MySQL 8.0. Cela nous permettra de stocker et d'archiver l'ensemble des données qui passeront à travers notre application. Cette base de données nous permettra de faire évoluer notre application future et permettra à l'application de fonctionner sans dysfonctionnement. Le serveur de base de données nous permet donc de stocker en masse nos données concernant notre application de vente de pizza. Nous optons pour MySQL car la communauté est grande, très active et cela nous permettra dans un court-terme d'élaborer une base de données sécurisée, robuste et ouverte à toute évolution. De plus, beaucoup de développeur savent écrire en SQL, ce qui permet une reprise assez facile (documentation ci-dessous).

Documentation : https://dev.mysql.com/doc/refman/8.0/en/

5.2 - Serveur Application WSGI

Gunicorn 20.0.4 :

« C'est un serveur de WSGI (cf. glossaire) en pur-python utilisé pour servir les applications python. Il a une interface utilisateur réfléchie, et est extrêmement facile à utiliser et à configurer. Gunicorn a des configurations par défaut saines et raisonnables. Cependant, certains autres serveurs, comme uWSGI, sont largement plus personnalisables, et par conséquent, sont beaucoup plus difficiles à utiliser efficacement. Gunicorn est le choix recommandé pour de nouvelles applications Python web aujourd'hui ». (Source et documentation ci-dessous)

Source : https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/web.html
Documentation : http://docs.gunicorn.org/en/latest/install.html

5.3 - Serveur web

Nginx 1.17.0 :

« Nginx (prononcé “engine-x ”) est un serveur web et un reverse-proxy pour HTTP, SMTP et d’autres protocoles. Il est connu pour sa haute performance, la simplicité relative, et sa compatibilité avec de nombreux serveurs d’applications (comme les serveurs WSGI). Il inclut aussi des fonctionnalités pratiques comme le load-balancing, l’authentification basique, le streaming, et d’autres encore. Conçu pour servir des sites à forte charge, Nginx est progressivement en train de devenir populaire. » (Source et documentation ci-dessous).

Source : https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/web.html
Documentation : https://nginx.org/en/docs/

5.4 - Framework Django

Django 3.0.3 : « Framework d’application web « tout en un », est un excellent choix pour la création de sites Web orientés contenus en python. »
(Source et documentation ci-dessous).

Source : https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/web.html
Documentation : https://docs.djangoproject.com/en/3.0/intro/overview/

6 - GLOSSAIRE

Composant	Un composant est une partie du logiciel de l'application qui aura des responsabilités précises. Il aura des interfaces qu'il peut proposer et d'autres interface où il fait appel à d'autres méthodes provenant des autres composants présents dans l'environnement de l'application.
Nœud	Dans un diagramme de déploiement, il représente soit une machine, soit un environnement d'exécution.
Protocole	TCP is connection-oriented, and a connection between client and server is established (passive open) before data can be sent. Une session TCP fonctionne en trois phases : l'établissement de la connexion ; les transferts de données ; la fin de la connexion. Source : https://en.wikipedia.org/wiki/Transmission_Control_Protocol
Serveur web Nginx	Nginx (prononcé "engine-x ") est un serveur web et un reverse-proxy pour HTTP, SMTP et d'autres protocoles. Il est connu pour sa haute performance, la simplicité relative, et sa compatibilité avec de nombreux serveurs d'applications (comme les serveurs WSGI). Documentation : https://nginx.org/en/docs/ Source : https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/web.html
WSGI	Le Web Server Gateway Interface (ou "WSGI" pour faire court) est une interface standard entre les serveurs Web et les frameworks web Python. En normalisant le comportement et la communication entre les serveurs Web et les frameworks web Python, WSGI permet d'écrire du code web Python portable qui peut être déployé dans tout serveur Web conforme à WSGI. Source : https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/web.html
Serveur WSGI (Gunicorn)	Gunicorn (Green Unicorn) est un serveur de WSGI en pur-python utilisé pour servir des applications Python. Contrairement à d'autres serveurs web Python, il a une interface utilisateur réfléchie, et est extrêmement facile à utiliser et à configurer. Gunicorn a des configurations par défaut saines et raisonnables. Documentation : http://docs.gunicorn.org/en/latest/install.html Source : https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/web.html
Framework	D'une manière générale, un framework web se compose d'un ensemble de bibliothèques et un gestionnaire principal au sein duquel vous pouvez construire un code personnalisé pour implémenter une application Web (par exemple, un site web interactif). Source : https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/web.html
Django	Framework d'application web « tout en un », est un excellent choix pour la création de sites Web orientés contenus. Documentation : https://docs.djangoproject.com/en/3.0/intro/overview/ Source : https://python-guide-pt-br.readthedocs.io/fr/latest/scenarios/web.html