



OC-Pizza

Pizza-Application

Dossier de conception technique

Version 1.0

Auteur

Le Magorou Jean-Martial

TABLE DES MATIÈRES

1 - Versions.....	3
2 - Introduction.....	4
2.1 - Objet du document.....	4
2.2 - Références.....	4
3 - Le domaine fonctionnel.....	5
3.1 - Référentiel.....	5
3.1.1 - Règles de gestion.....	6
3.1.1.1 - Classes périphériques.....	7
3.1.1.2 - Classe/table Client.....	7
3.1.1.3 - Classe/table Restaurant.....	7
3.1.1.3.1 Relations.....	7
3.1.1.4 - Classe/table User.....	8
3.1.1.4.1 Relations.....	8
3.1.1.5 - Classe/table Address.....	8
3.1.1.5.1 Relations.....	8
3.1.2 - Classes Centrales.....	9
3.1.2.1 - Classe/table OrderPizza.....	9
3.1.2.1.1 Relations.....	9
3.1.2.2 - Classes/tables Pizza, Ingredient et AdditionalProduct.....	11
3.1.2.2.1 Relations.....	11
3.1.3 - Nota Bene.....	12
3.1.3.1 - Les tables d'énumération.....	12
3.1.3.2 - Le mot de passe.....	12
4 - Architecture Technique.....	13
4.1 - L'application WEB.....	13
4.2 - La base de données.....	14
4.3 - Les Composants externes.....	14
4.3.1 - Google maps – Géolocalisation.....	14
4.3.2 - Braintree – Règlement par internet.....	15
4.3.3 - Nexmo – Notification des clients par SMS.....	17
5 - Architecture de Déploiement.....	18
5.1 - Les Matériels utilisateurs.....	18
5.2 - La plateforme.....	18
6 - Architecture logicielle.....	20
6.1 - Principes généraux.....	20
6.1.1 - Les couches.....	20
6.1.2 - Structure des sources.....	20
7 - Points particuliers.....	22
7.1 - Ressources.....	22
7.2 - APIs.....	22
7.3 - Développement.....	22
7.4 - Packaging/Livraison.....	23
8 - Glossaire.....	23

1 - VERSIONS

Auteur	Date	Description	Version
JMLM	10/05/20	Création – domaine fonctionnel	V0.1
JMLM	02/08/20	Architectures	V0.2
JMLM	03/08/20	Points particuliers – Glossaire	V0.3
JMLM	05/08/20	Correction	V 0,4

2 - INTRODUCTION

2.1 - Objet du document

Le présent document constitue le volet « conception technique » de la réponse à l'appel d'offre qu'a fait la société OC-Pizza afin de remplacer son Système d'Information actuel. Ce document découle du précédent document « Dossier des spécifications techniques ».

Il a pour but de présenter les différents acteurs et fonctionnalités afin de :

- Modéliser les objets du domaine fonctionnel et ainsi définir le modèle physique de données qui permettra de concevoir la base de données de ce qui sera le futur S. I. d'OC-Pizza.
- Définir les composants internes et externes afin de définir leurs interactions avec le S. I. d'OC-Pizza.
- Décrire le déploiement des composants
- Décrire l'architecture logicielle de l'application ainsi que les modules nécessaires à sa bonne exécution.

Les éléments du présent dossiers découlent :

- de l'expression du besoin (cf mail du 14/04/2020)
- de différents entretiens passés avec des collaborateurs OC-Pizza en date du 28/04/2020
- de l'analyse complète des besoins suite a ces entretiens

2.2 - Références

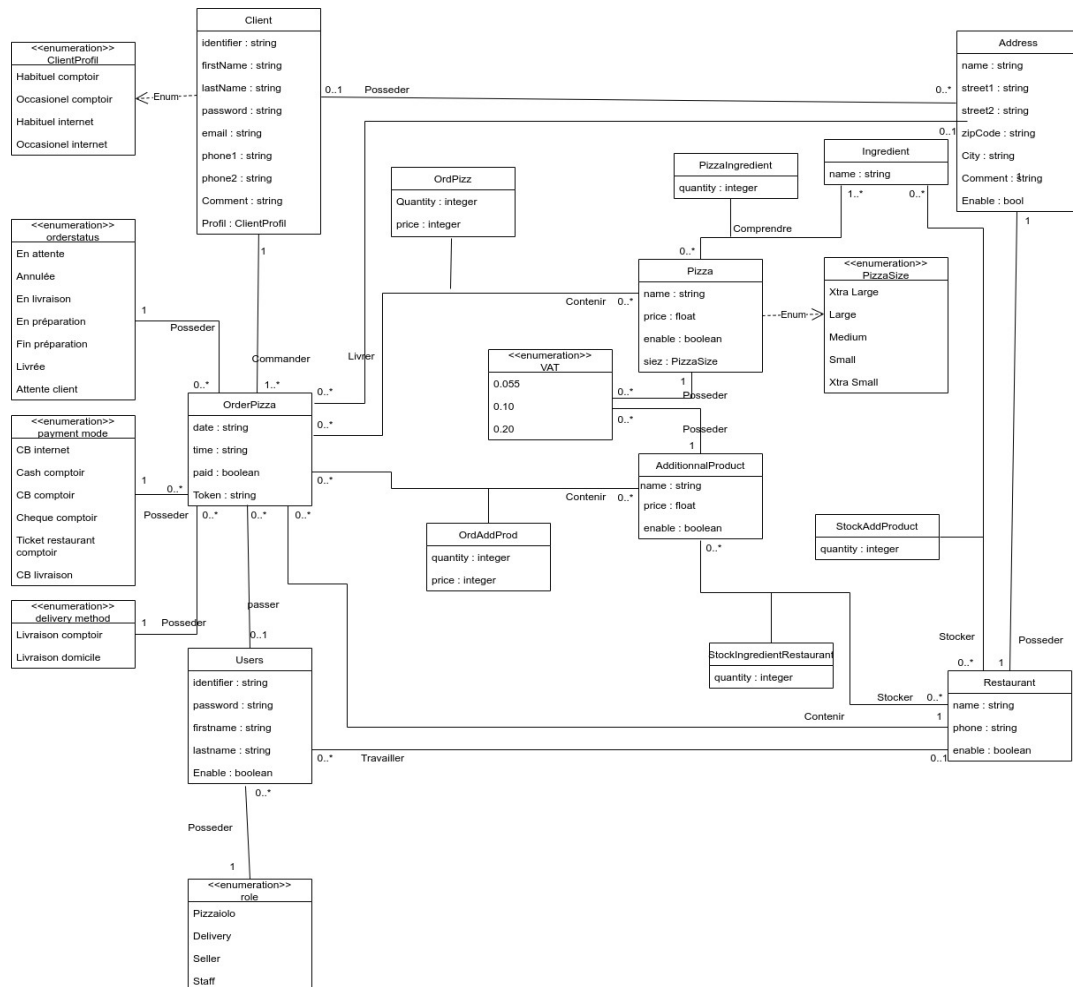
Pour de plus amples informations, se référer également aux éléments suivants :

1. **P9 – Dossier Conception Fonctionnelle** : Dossier de conception fonctionnelle de l'application
2. **P9 – Dossier Exploitation** : Dossier d'exploitation de l'application
3. **P9 – PV Livraison** : PV de livraison de l'application

3 - LE DOMAINE FONCTIONNEL

3.1 - Référentiel

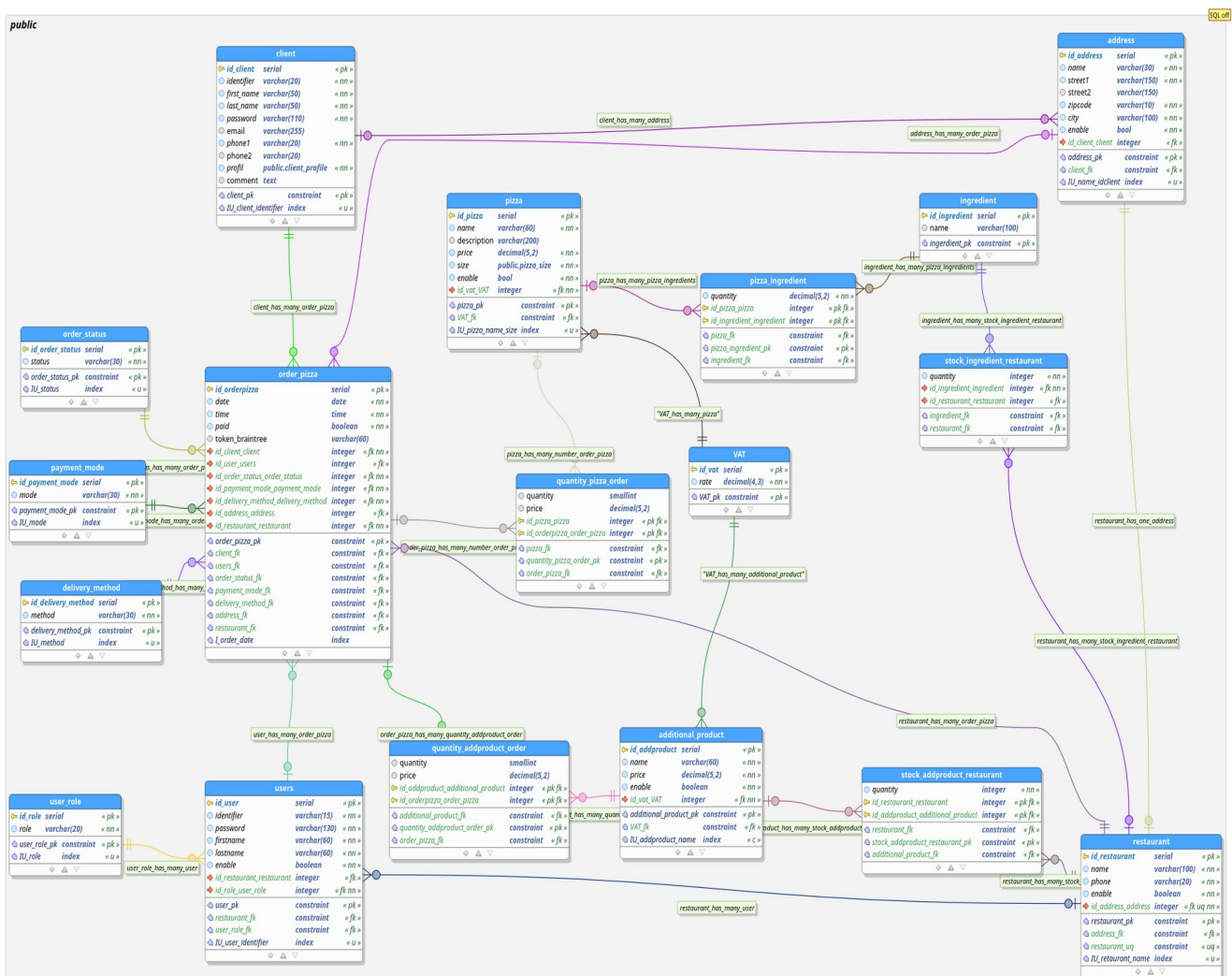
Le domaine fonctionnel permet d'établir les différentes classes interférant dans le S.I. ainsi que les relations qui les lient. Le premier schéma est appelé le diagramme de classes. Il servira de socle à la création du Modèle physique de données et donc à la création de la base de données.



Le diagramme de classes UML ci-dessus fournit une représentation précise des informations manipulées et permet ainsi d'avoir une vision globale du système et de l'organisation des données.

Du diagramme de classes découle un second diagramme plus détaillé qui est la modélisation exacte de la base de données : le Modèle physique de données.

Nous décrirons ensuite chaque table/classe, les différentes colonnes/attributs qui les composent ainsi que les liens/associations qui les relient entre-elles.



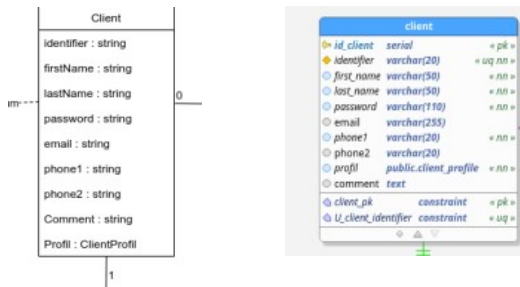
3.1.1 - Règles de gestion

Nous allons voir que les classes et les tables sont étroitement liées, car les secondes émanent des premières. Dans la section suivante nous allons décrire les classes/tables ainsi que les liens qui les relient entre-elles. Les classes ont été classées en 2 groupes : Les classes centrales qui concernent directement le cœur de métier, c'est-à-dire les commandes et les ventes, puis les classes périphériques qui viennent alimenter les classes centrales. Vous pouvez voir sur les 2 diagrammes précédents que les classes centrales sont au centre du schéma et sont entourées par les classes périphériques. Pour une meilleure compréhension nous commencerons par décrire les classes périphériques.

Dans les descriptifs qui vont suivre, quand nous parlerons de cardinalité et pour une meilleure compréhension, nous partirons toujours de la classe/table dont nous faisons la description. Ce sera donc toujours la table qui contiendra la clé étrangère (ou foreign key) notée *nomtable_fk* (ex : address_fk).

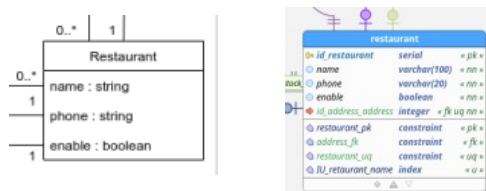
3.1.1.1 - Classes périphériques

3.1.1.2 - Classe/table Client



La Classe client décrit le client en stockant les Nom – Prénom, identifiant – mot de passe, mail, 2 numéros de téléphone dont un seul est obligatoire et éventuellement un commentaire. Il a aussi été mis un profil qui peut être Occasionnel ou Habitué en vue d’offres commerciales ou de campagnes publicitaires.

À noter que le client est répertorié dans la base par un identifiant unique mais qu’il est tout à fait possible d’avoir 2 clients avec le même nom-prénom ou même 2 clients à la même adresse.



3.1.1.3 - Classe/table Restaurant

La classe Restaurant va contenir les coordonnées d’un restaurant. Toutefois, seuls les noms, téléphone et un statut a enable/disable sont réellement enregistrés dans la table. L’adresse et les employés qui y sont rattachés le sont par des liens avec d’autres classes. Le statut enable/disable permettra de faire apparaître ou non le restaurant sur le site (possibilité de créer un restaurant avant son ouverture).

3.1.1.3.1 Relations

Classe : Address – Cardinalité : de 0,1 à 1

La classe Restaurant est reliée à la classe Adress. Toutefois, un restaurant ne peut et ne doit avoir qu’une et unique seule adresse alors qu’une adresse va correspondre à 1 restaurant ou à un client.

3.1.1.4 - Classe/table User



La classe User, comme dit ci-dessus, comprend tous les utilisateurs/employés d'OC-Pizza. Elle contient un identifiant unique ainsi que le mot de passe crypté de l'utilisateur. Un petit Nota Bene sur le mot de passe se trouve à la fin de ce chapitre. Cette classe contient aussi le rôle de chacun ainsi que les noms et prénoms.

3.1.1.4.1 Relations

Classe : Restaurant – Cardinalité : 0,n à 1

Un employé ne peut être rattaché qu'à un seul restaurant. Toutefois, le Staff peut et ne doit pas être rattaché à un point de vente. Enfin, un restaurant va avoir de 0 (en cours de création) à N employés.

Classe : Role – Cardinalité : 0,1 à 0,n

Un employé à un rôle (Staff, Pizzaiolo...). Ces rôles ne sont pas bloqués, il sera toujours possible d'en rajouter facilement par la suite.

3.1.1.5 - Classe/table Address



La classe address contient toutes les adresse que ce soit celles d'un client ou d'un restaurant. Une adresse reste une adresse. Aux champs classiques d'une adresse tels que la rue, le code postal, la ville... a été rajouté un état (enable) que l'on peut mettre à Vrai ou Faux. Cela permettra aux clients de supprimer une de leur adresse sans que cela la supprime vraiment afin de pouvoir conserver une ancienne commande à cette adresse intacte et de pouvoir ainsi la ressortir si besoin.

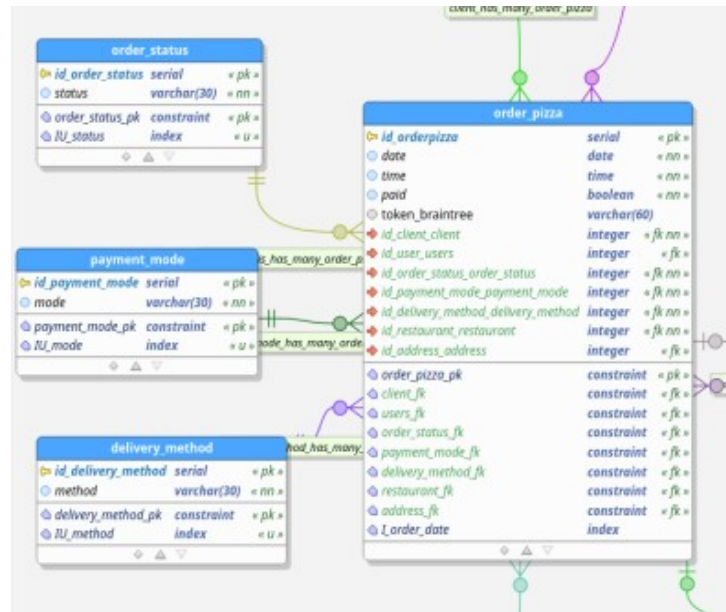
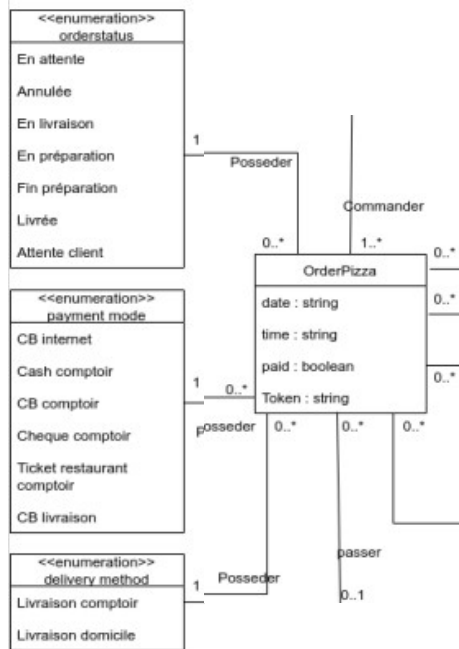
3.1.1.5.1 Relations

Classe : Client – Cardinalité : 0,n à 0,1

A une adresse correspond de 0 (si c'est un restaurant) a 1 client. Ne pas oublier que l'on peut avoir plusieurs fois la même adresse pour des clients différents (Vous avez la même adresse que votre voisin de palier quand vous habitez en immeuble!). Un client peut avoir plusieurs adresses sachant qu'il ne s'agit là que d'adresse de livraison (domicile, travail...).

3.1.2 - Classes Centrales

3.1.2.1 - Classe/table OrderPizza



Cette classe est la première des classes dites centrales. Nous pourrions presque dire que c'est la classe du centre des classes centrales, car elle est directement ou indirectement reliée à toutes les autres. En effet, si en tant que donnée propre elle ne contient qu'une date et heure, un statut de règlement (attribut paid) à oui ou non ainsi qu'un élément de paiement si le règlement s'est effectué par le site Internet que l'on appelle un Token et dont nous parlerons plus tard, cette classe contient finalement via ses liens tout ce qui concerne un bon de commande. Tout d'abord, abordons les liens qui ne concernent pas les produits mais plutôt la commande proprement dite. Il s'agit de 3 liens avec des classes d'énumération que je détaillerai plus loin. Ces classes sont OrderStatus (statut de la commande), PaymentMode (mode de règlement) et Delivery(Méthode). Pour ces 3 classes les cardinalités des liens sont les mêmes(1 à 0,n) car une commande a obligatoirement dès sa création un statu, un mode règlement et un mode de livraison (qui peut aussi être « retrait comptoir »).

3.1.2.1.1 Relations

Classe : Client – Cardinalité : 0,n à 1

Une commande n'est qu'à un seul et unique client. À l'inverse, un client peut bien entendu avoir plusieurs commandes (cf classe Client).

Classe address – Cardinalité : 0,n à 1

Il s'agit là d'une adresse de livraison. Si un client a opté pour le retrait au comptoir de sa commande, il n'y a pas d'adresse. Enfin une commande ne pourra pas avoir plusieurs adresses de livraison.

Classe : Users – Cardinalité : 0,n à 1

Une commande si elle est passée par le client directement par le site n'aura pas d'utilisateur OC-Pizza associé tout de suite. Par contre, au fur et à mesure de l'avancée de la commande l'utilisateur qui lui sera associée changera tout comme le status.

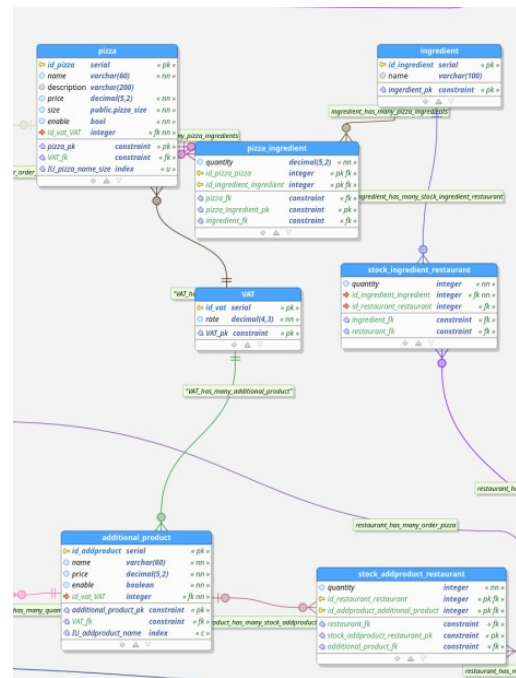
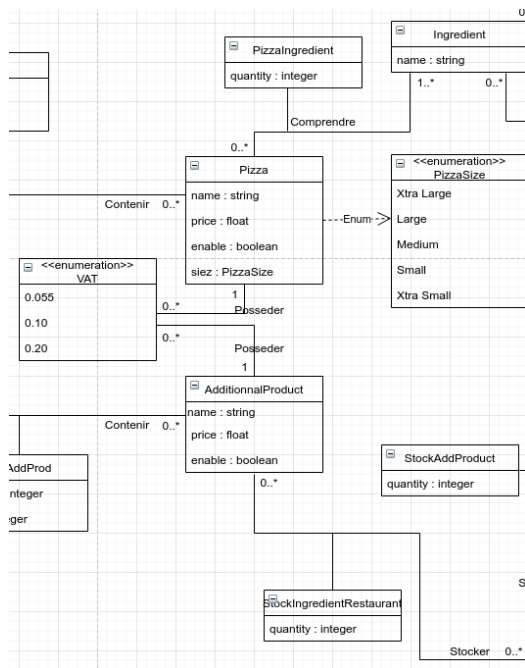
Classe : Restaurant : Cardinalité : 0,n à 0,1

Comme dit précédemment une commande appartient à un seul et unique restaurant qui la traitera. Par contre, un point de vente en cours de création n'aura pas (encore) de commandes qui lui seront attribuées

Classes : Pizza et AdditionalProduct – Cardinalité : 0,n à 0,n

Ces 2 classes sont complètement similaires et peuvent être traitées de la même manière. Une commande peut contenir de 0 à plusieurs pizzas et/ou de 0 à plusieurs produits additionnels. Il peut paraître bizarre de ne pas vendre de pizza mais si un client se présente au comptoir et veut par exemple un muffin et une boisson gazeuse au cola, il ne faut pas lui refuser... Inversement une pizza ou un produit additionnel pourra apparaître sur 0 ou plusieurs commandes. Les liens de plusieurs à plusieurs ont une spécificité, ils nécessitent de passer par une classe/table intermédiaire qui contiendra les clés de chacune des tables liées. Ces tables intermédiaires peuvent contenir d'autres attributs (colonnes) afin d'enrichir la relation. C'est le cas ici avec une quantité mais aussi un prix. Ce prix permettra d'avoir le tarif du produit au moment de la commande sans qu'il ne bouge dans le temps. Vous pourrez donc ressortir une commande/facture bien plus tard avec la quantité et le tarif du moment.

3.1.2.2 - Classes/tables Pizza, Ingredient et AdditionalProduct



Les classes Pizza et AdditionalProduct sont tellement similaires qu'elles peuvent être décrites en même temps. Elles sont toutes les 2 composées d'un nom de produit, d'un prix (le prix à l'instant T) et d'un « flag » qui permet de les désactiver et donc de ne pas les montrer sur le site.

Attention, le manque d'ingrédients sera traité différemment, mais vous pourriez vouloir mettre au point une recette et d'ores et déjà avoir créé la pizza. La classe pizza dispose en plus d'un attribut spécifique pour la taille en énumération (Xtra Large, Large, Medium, Small, Xtra Small).

La classe Pizza est reliée à une classe ingrédients (pour les recettes) puis les ingrédients a la classe Restaurant alors que la classe AdditionnalProduct est relié à la classe Restaurant en direct.

Chacune d'entre elle est aussi reliée à une classe VAT (TVA). La politique de la TVA française peut être changeante et en cas de modification/ajout d'un taux de TVA, il sera plus simple de le faire sur une table annexe.

Pour ce qui est de la classe Ingrédient, Elle est bien entendu inséparable de la classe pizza. C'est une table qui sert un peu de pivot entre le point de vente et la pizza pour les recettes d'un côté et pour les stocks de l'autre.

3.1.2.2.1 Relations

Classe Pizza

Classe : Ingrédient - Cardinalité : 0,n à 0,n

Il est évident qu'il faut plusieurs ingrédients pour faire une pizza et qu'un même ingrédient sert dans plusieurs pizzas. Par contre comme je vous l'avais dit précédemment, une relation de plusieurs à plusieurs comme c'est le cas ici nécessite une table supplémentaire (pizza_ingredient). Cette table contiendra outre les clés pizza/ingrédient les quantités nécessaires. La liaison pizza-quantité-ingrédient sera ainsi complète.

Classe : VAT - Cardinalité : 0,n à 1

A une Pizza ne correspond qu'un taux de TVA. Par contre le taux de TVA 20 % n'aura pas de correspondance dans la table pizza alors que le taux 5,5 % en aura plusieurs.

Classe AdditionalProduct

Classe : Restaurant – Cardinalité 0,n à 0,n

Comme il a déjà été dit, la cardinalité plusieurs à plusieurs nécessite la création d'une table « tampon ». Cette table tampon (stock_addproduct_restaurant) aura donc en plus des clés addproduct et restaurant une quantité (stock) que sera en fait le stock de chaque produit additionnel pour chaque point de vente.

Classe : VAT – Cardinalité 0,n à 1

La relation est la même que pour la classe Pizza

Classe Ingredient

Classe : Restaurant – Cardinalité : 0,n à 0,n

Nous sommes en présence d'exactly la même relation que AdditionalProduct/Restaurant : Une relation de plusieurs à plusieurs avec le stock au milieu ! Nom de la table « tampon » : stock_ingredient_restaurant.

3.1.3 - Nota Bene

3.1.3.1 - Les tables d'énumération

Ces tables sont au nombre de 3 et sont en fait des énumérations. Le choix de Postgresql en tant que moteur de base de données a conduit à faire des tables plutôt que des énumérations directes dans la table order_product pour plusieurs raisons :

- Performance : PostgreSQL pour un type Enum crée en fait une table interne appelé Type. Pour accéder à une énumération un accès table est donc effectué quel que soit le mode choisi (table ou enum).
- Les énumérations dont il est question ici vont se retrouver dans l'application sous forme de select-box à plusieurs endroits. Si vous souhaitez rajouter un statut afin de suivre plus finement le workflow de la commande (par exemple : Cuisson), il faudra modifier tous les select box dans toute l'application, ce qui peut vite devenir fastidieux. La modification d'un libellé est aussi très fastidieuse à faire.

3.1.3.2 - Le mot de passe

Concernant le mot de passe, je voudrais insister sur le fait qu'il est crypté directement par une librairie python (crypt en mode SHA256) et que si sa longueur en texte est de maximum 30 caractères alphanumériques, le mot de passe crypté fait plus de 100 caractères de long. De plus, il est impossible de récupérer un mot de passe perdu, il faudra en recréer un !

4 - ARCHITECTURE TECHNIQUE

L'application sera une application de type web elle-même basée sur plusieurs briques. Ces briques mises côte à côte forment le serveur d'application. C'est cette application qui communiquera avec les utilisateurs d'un côté et avec les composants externes de l'autre

Les composants externes sont décrits par les diagrammes de composants. Ceux-ci mettent en évidence les dépendances entre eux et les composants internes et décrivent les interfaces nécessaires à leur utilisation.

Ces composants sont au nombre de 3 : Google Maps pour la géolocalisation, Braintree pour le paiement via le site internet ou le téléphone du livreur et Nexmo pour les envois de SMS au client.

4.1 - L'application WEB

L'application web sera une application web responsive. Cela veut dire que cette application pourra être utilisée sur différents médias (PC, tablette, smartphone). Cette application devra être développée sur une plateforme qui permette une évolution simple à mettre en œuvre tout en restant robuste et résistante à la charge.

L'application sert les 3 interfaces décrites dans le cahier des charges fonctionnel. Le serveur sera donc le serveur frontal quel que soit le type d'utilisateur et quelle que soit la demande de l'utilisateur.

L'application sera composée des briques logicielles suivantes :

- Système d'Exploitation : Linux Ubuntu 20.04 (Sortie Avril 2020)

La distribution Linux Ubuntu existe maintenant depuis plus de 15 ans. Cette distribution est très utilisée tant pour les postes de travail que pour les serveurs. Les versions X.04 sont des versions Long Time Support et vous garantissent un support gratuit de la part de l'éditeur de 5 ans.

- Langage utilisé : Python 3.8 (sortie Octobre 2019)

Le langage python est depuis quelques années déjà l'un des langages les plus utilisés. Il est reconnu pour sa rapidité tant au niveau de l'exécution et de développement ainsi que pour le grand nombre de bibliothèques externes qui lui offrent un grand nombre de fonctionnalités.

- Framework + ORM : Django 3.0 (sortie Décembre 2019)

L'utilisation d'un framework tel que Django couplé à des bibliothèques telles que Bootstrap et JQuery donne des applications « full responsive » avec une excellente ergonomie. De plus, l'utilisation d'un framework web permet un gain de temps en termes de développement. Cela permet aussi une organisation du code source standardisée qui est donc facilement maintenable.

- Serveur Web : Apache 2.4 (sortie Août 2019)

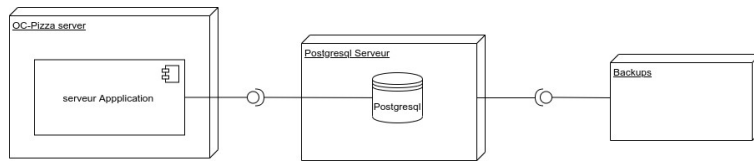
Le serveur web apache existe depuis 1995. Sa réputation n'est donc plus à faire et sa robustesse est bien connue dans le monde du Web. De plus, tous les modules, bibliothèques ou autres fonctionnent sous Apache ce qui n'est pas toujours le cas de ses concurrents.

- Serveur de base de données : Postgresql 12 (sortie Juin 2019)

Là encore un « dinosaure » de 1996 et qui a eu le temps de démontrer toutes ses qualités. La donnée est le cœur du système. Il n'est pas concevable de confier ses données à un serveur qui n'a pas toutes les qualités requises : Rapidité, fiabilité, sécurité. Postgresql rassemble toutes ses

qualités et dispose aussi d'outils avancés tels que de la réplication en temps réel par exemple.

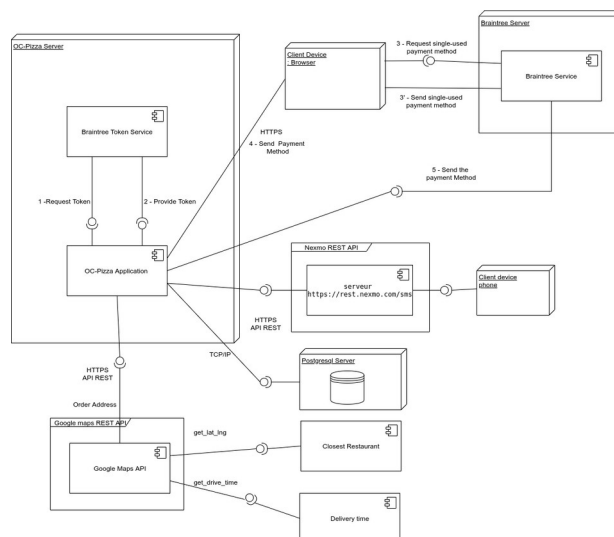
4.2 - La base de données



La base de données sera donc un serveur Postgresql. Le système d'exploitation de ce serveur sera Linux. Les sauvegardes et réorganisations de la base de données seront effectuées de nuit en dehors des heures d'ouverture.

4.3 - Les Composants externes

Le diagramme suivant montre les différents composants externes nécessaires à l'application.



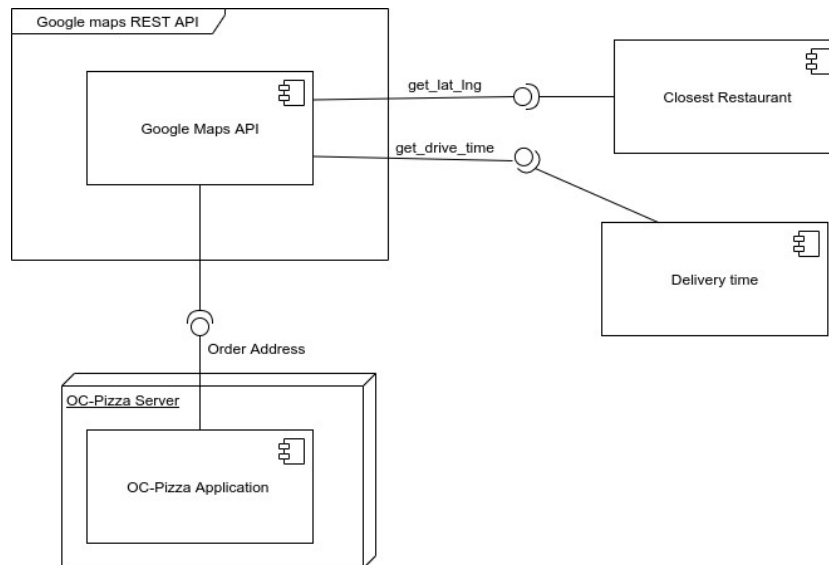
Nous les détaillerons un par un ensuite


4.3.1 - Google maps – Géolocalisation

Le serveur OC-Pizza fournit à l'API Google Maps l'adresse de livraison. L'API Google fournit le temps de parcours et le meilleur itinéraire pour la livraison ainsi que le restaurant le plus proche de la position de l'utilisateur s'il le souhaite.

Les tarifs de Google Maps ont évolué mi-2018, mais le quota de gratuité ne devrait pas être atteint.

Le diagramme de composant ci-dessous décrit ce principe



Pour tout ce qui est de l'affichage de la carte sur le site avec  le petit logo bien connu pour la position des restaurants, la carte s'obtient directement sans faire appel aux APIS.

Le guidage du livreur se fera au travers d'un GPS Google maps ou autre tel que ceux que nous utilisons tous les jours.

4.3.2 - Braintree – Règlement par internet

Le règlement par internet doit répondre à des normes de sécurité et de confidentialité très importantes. Seuls quelques acteurs proposent une offre répondant à ces obligations tout en mettant à la disposition de leurs clients des outils de suivi, d'anti fraude évolués et la possibilité d'avoir plusieurs moyens de paiement tels que les cartes VISA et MasterCard, PayPal, Applepay ou Googlepay, bitcoins...

L'API Braintree nécessite de mettre en œuvre un dialogue à trois (client – serveur OC-Pizza – serveur Braintree) plus complexe que les 2 autres composants externes. Cet échange permet de garantir que la transaction correspond bien aux propriétés ACID et qu'elle s'est donc bien exécutée.

ACID : Atomicité – Cohérence – Isolation - Durabilité

Atomicité : Garantie que la transaction soit faite complètement ou pas du tout.

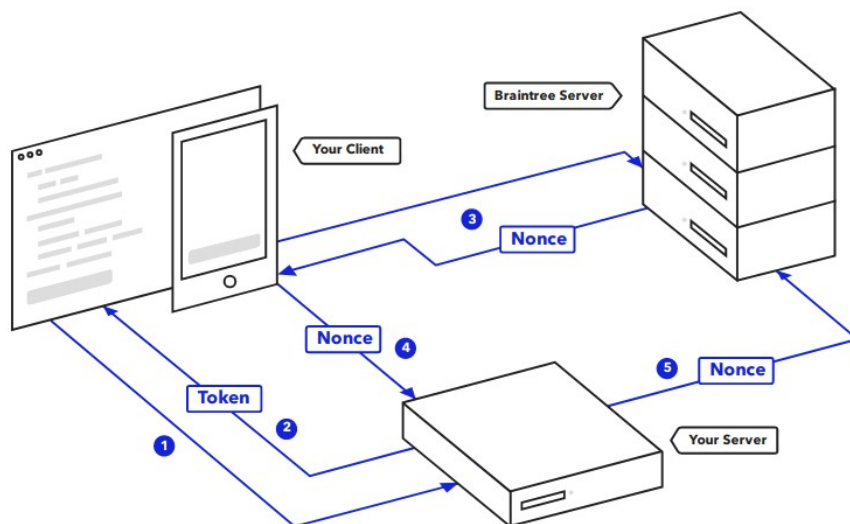
Cohérence : L'état du système avant et après la transaction est valide

Isolation : La transaction ne dépend d'aucune autre.

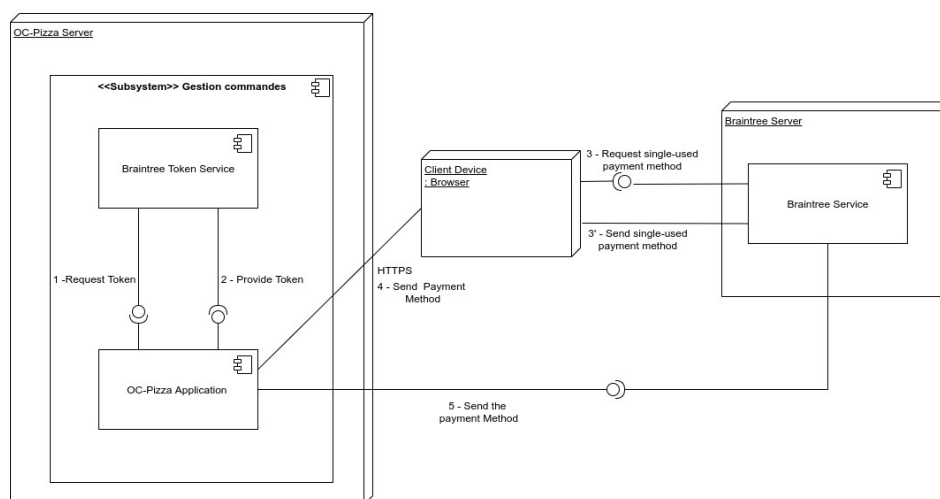
Durabilité : Garantie que la transaction a bien été enregistrée de façon permanente.

Il va de soi qu'une transaction bancaire se doit de répondre à ces critères et que le fait de passer par un acteur bien connu du marché garanti cela.

Le schéma suivant fourni par Braintree nous montre le workflow du règlement.



Le diagramme de composant montre là encore le principe de fonctionnement :



Étape 1 : Le serveur frontal d'application OC-Pizza demande un jeton client à votre serveur et initialise le SDK client.

Étape 2 : Le serveur OC-Pizza à travers un programme (service) fourni par Braintree génère et renvoie un jeton client à votre client à l'aide du SDK du serveur.

Étape 3 : Le client d'OC-Pizza soumet des informations de paiement, le SDK du client communique ses informations à Braintree et renvoie un nonce de méthode de paiement fourni par le serveur Braintree.

Étape 4 : Le client OC-Pizza renvoie le mode de paiement nonce au serveur d'application OC-Pizza.

Étape 5 : Le serveur OC-Pizza utilise le SDK du serveur pour créer une transaction et l'envoyer au serveur Braintree afin de valider la transaction.

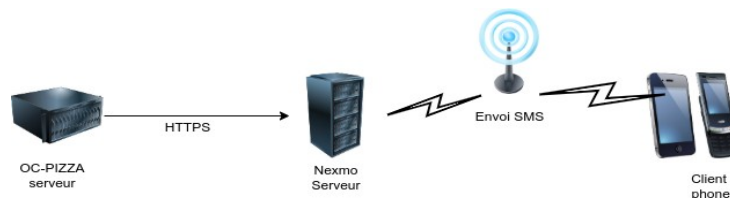
Ce système permet donc de sécuriser la transaction de bout en bout et dispose d'un système anti-fraude élaboré.

L'implémentation de Braintree dans Django se fait aisément. La partie côté client est en JavaScript et la partie côté serveur se fait bien sûr en Python. L'API est largement documentée et Braintree met à disposition un SDK avec de nombreux exemples.

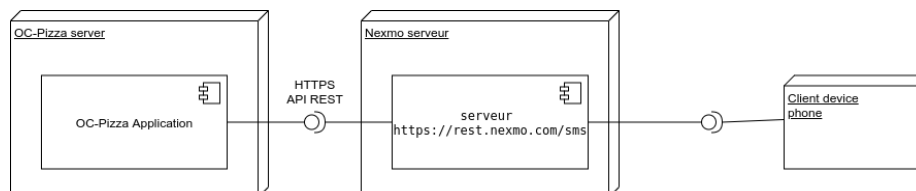
4.3.3 - Nexmo – Notification des clients par SMS

Cette interface est plus simple que les 2 précédentes. Dans un souci de simplification et de simplicité nous nous contenterons d'envoyer le SMS sans attendre d'accusé réception ou quoi que ce soit d'autre. Il s'agit là, juste d'un envoi. Par contre, il est important de pouvoir prévenir les clients qui ont opté pour le retrait en point de vente que leur pizza est prête. Il faudra toutefois être vigilant aux tarifs et ne pas hésiter à changer de prestataire si besoin, car la simplicité de cette API permet une grande souplesse pour cela.

L'envoi d'un SMS se fait via le protocole HTTPS en envoyant une requête formatée sur le serveur de Nexmo. Ce même serveur se chargera ensuite d'envoyer le SMS.



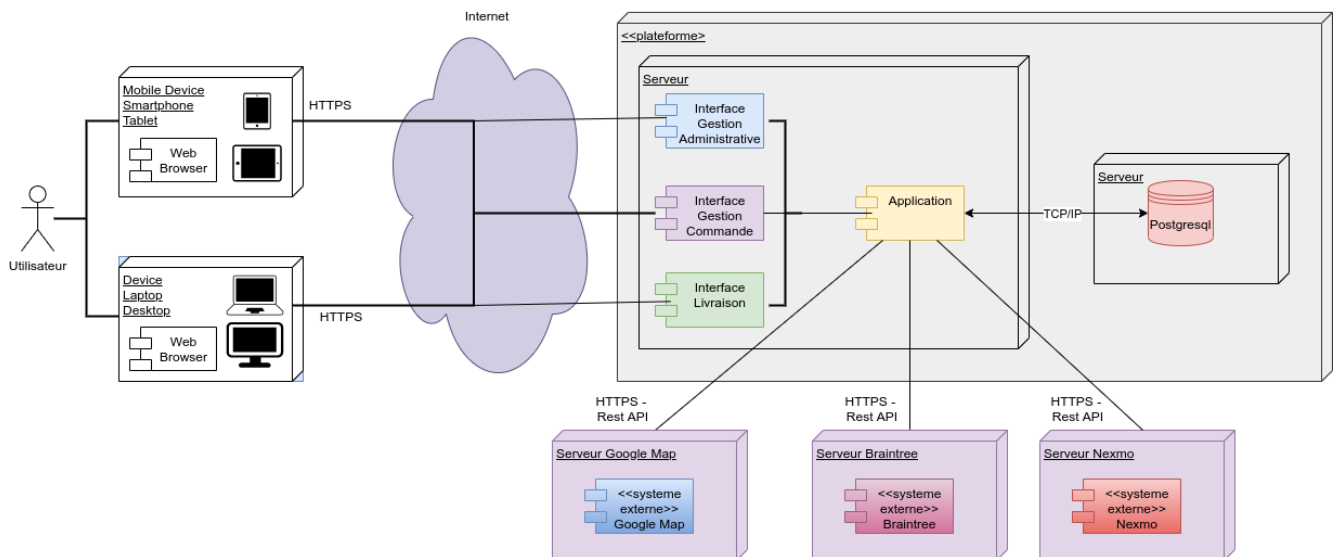
Le diagramme de composant est là aussi relativement simple et parle de lui-même.



Nous voyons donc que les interfaces ne sont que dans un sens. Il n'y a donc aucun retour attendu de l'envoi du SMS.

5 - ARCHITECTURE DE DÉPLOIEMENT

Après avoir vu chaque composant, le diagramme de déploiement UML permet aisément d'identifier sur quel matériel physique se place chaque composant ainsi que les différentes connexions qui les relient. Il permet donc de bien comprendre comment sera déployée l'application, et par quel moyen chaque acteur primaire ou secondaire devra y accéder.



On peut donc voir sur le graphique que l'architecture est découpable en 3 grandes parties :

- Les matériels utilisateurs
- La plateforme
- Les acteurs secondaires

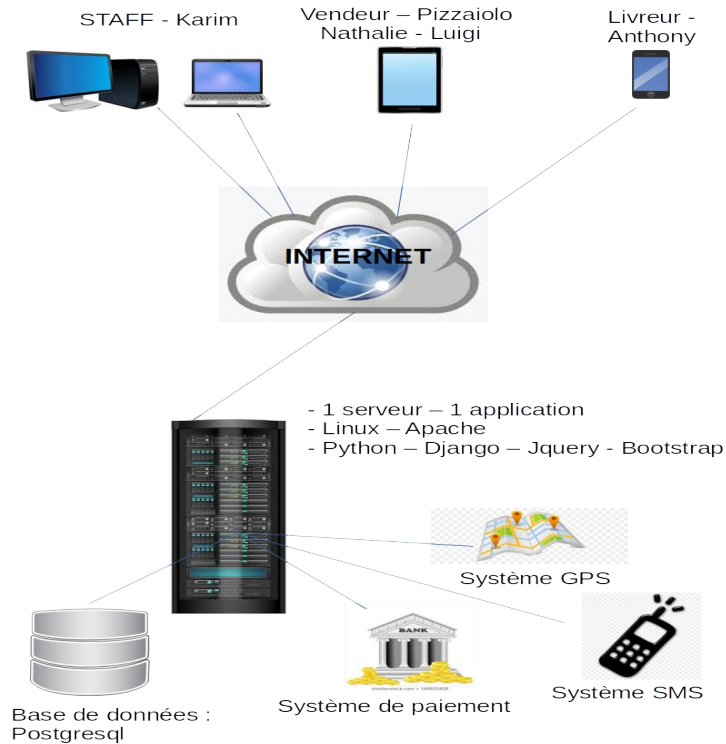
5.1 - Les Matériels utilisateurs

Quels que soient les utilisateurs (clients, employés OC-Pizza) l'application est la même. Les matériels utilisés par contre peuvent différer. Cela peut aller du simple smartphone au PC avec un grand écran. Il suffira de disposer d'un simple accès à Internet et d'un navigateur standard. Il est évident que les matériels utilisés par les employés d'OC-Pizza seront différents. Les vendeurs pourront avoir un ordinateur avec un clavier et un écran tactile pas trop grand afin de saisir facilement une commande par téléphone et pour pouvoir avoir une interactivité forte avec le logiciel. Les pizzaiolos pourront eux utiliser un écran tactile type tablette, car ils n'auront rien à taper au clavier. Les livreurs devront eux utiliser un smartphone pour des simples questions d'encombrement. Quant au staff, il devra utiliser un écran plus grand pour pouvoir avoir un confort de lecture des tableaux de bord.

5.2 - La plateforme

L'application OC-Pizza sera déployée sur un serveur VPS de la société OVH. Cet hébergeur est un

des pionniers de l'hébergement web et dispose d'une offre non seulement importante mais surtout évolutive en termes de puissance et de volume. Cet hébergeur permet sur une seule offre de disposer du nom de domaine, éventuellement d'un serveur mails hébergé mais aussi d'une base de données postgresQL en cloud sauvegardée et sécurisée. De plus, les tarifs pratiqués sont très concurrentiels. Le fait de disposer de votre propre serveur applicatif vous permettra de faire évoluer votre SI à votre gré et en toute indépendance.



django

 **python™**

B

jQuery
write less, do more.



Google Maps



6 - ARCHITECTURE LOGICIELLE

6.1 - Principes généraux

Les sources et versions du projet sont gérées par **Git**, les dépendances sont-elles gérées par **pip**.

L'application se présentera donc sous la forme d'un projet Django divisé en 3 parties ou applications :

- app_admin : Gestion administrative
- app-commandes : Gestion des commandes
- app_livraisons : Gestion des livraisons

Ce mode de fonctionnement en parties permet de sortir relativement facilement une partie pour la réutiliser ailleurs (Gestion administrative par exemple) mais aussi de rajouter une partie (commercial ou comptabilité par exemple) sans avoir à tout refaire.

Chaque partie respecte le pattern Model-View-Template. Ce modèle est inhérent à Django et est suivi maintenant par une très grande majorité de développement et d'outils.

6.1.1 - Les couches

Comme dit ci-dessus, L'architecture applicative est du type MVT:

- La couche **Model** : Responsable de la représentation des données. Pour schématiser, elle va représenter les données et les fonctions (méthodes) qui leur sont liées.
- La couche **Template** : Responsable de l'affichage des données.
- La couche **Vue** : Responsable de l'interface client c'est en fait la logique du programme qui est géré par cette couche. Elle reçoit des demandes (requêtes) demande si besoin à la couche **model** de lui renvoyer des données et les renvoie une fois traitées à la couche **template** pour qu'elle les affiche correctement. J'ai préféré expliquer cette couche en dernier, mais on voit bien qu'elle se situe au milieu des 2 autres.

Exemple pour l'application app_commandes :

- La couche **Model** va représenter les données : classes pizza_ingredients, stocks...et les fonctions liées comme par exemple ingrédient_en_stock...
- La couche **Vue** va être la logique : Un client arrive sur le site, la couche **model** reçoit une demande d'affichage de la page de la liste des pizzas. Elle va donc demander à la couche Model la liste des pizzas dont les ingrédients sont en stock et va envoyer ces données filtrées et classées à la couche Template
- La couche **Template** va afficher les données reçues en respectant les chartes graphiques, les taille d'écran...

6.1.2 - Structure des sources

La structuration des répertoires du projet suit la logique suivante :

- Les répertoires et fichiers ci-dessous ne sont indiqués qu'à titre d'exemple. L'ordre n'est pas respecté et d'autres seront créés au fur et à mesure du développement et des besoins.

```

oc_pizza
├── app_admin                                     (ici 3 applications donc 3x)
│   ├── admin.py
│   ├── apps.py
│   ├── forms.py
│   ├── __init__.py
│   ├── migrations
│   ├── models.py
│   ├── static
│   │   └── app_admin
│   │       ├── css
│   │       │   └── style.css
│   │       ├── img
│   │       │   └── image.jpeg
│   │       ├── js
│   │       │   └── script.js
│   ├── templates
│   │   └── app_admin
│   │       ├── home.html
│   │       └── mentions.html
│   ├── urls.py
│   └── views.py
├── manage.py
├── oc_pizza
│   ├── asgi.py
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── README.md
├── requirements.txt
├── settings.py
├── staticfiles
├── static
│   └── img
│       ├── favicon.png
│       └── logo
└── templates
    ├── errors
    │   └── errors.html
    └── layouts
        ├── base.html
        └── partials
            ├── _back.html
            ├── _footer.html
            ├── _nav.html
            ├── _paginator.html
            └── _scripts.html

```

7 - POINTS PARTICULIERS

7.1 - Ressources

Les ressources graphiques telles que logo, dessins, polices, couleurs, images et photos... seront fournies par OC-Pizza au plus tard à la date de début du développement. Elles devront être dans un format et d'une résolution directement exploitables.

Les wireframes (esquisses) des pages du site devront être terminées au début du développement.

Un collaborateur de Pharos représentera OC-Pizza au sein de l'équipe de développement afin de perturber le moins possible le fonctionnement d'OC-Pizza lors de la phase de développement. Toutefois, un temps de disponibilité, d'une personne désignée d'OC-Pizza sera nécessaire épisodiquement afin de valider ou compléter certains détails.

7.2 - APIs

L'utilisation de l'API Google Maps nécessite une clé et l'ouverture d'un compte Google. OC-Pizza devra se munir d'une clé disponible à cet endroit :

<https://developers.google.com/maps/documentation/directions/get-api-key>. Cette clé est gratuite mais doit appartenir à l'exploitant du site.

OC-Pizza devra acquérir la solution de paiement Braintree au plus tard à la date de début du développement. Cette solution sera la solution de paiement en ligne et fera l'objet d'une grande vigilance dans les tests et validation. La solution Braintree peut s'acquérir via ce lien : <https://www.braintreepayments.com/fr/braintree-pricing>

L'utilisation de l'API Nexmo (SMS) nécessite elle aussi l'acquisition d'une clé secrète. OC-Pizza devra faire l'acquisition d'une telle clé et la fournir au collaborateur de Pharos chargé de le représenter durant toute la phase de développement. Cette clé peut être acquise via ce lien : <https://dashboard.nexmo.com/sign-up>

Pour l'acquisition de ces clés, nous nous proposons bien entendu de vous assister gracieusement.

7.3 - Développement

Le développement de l'application se fera dans les locaux de Pharos consulting. Aucun outil particulier, mises à part les clés d'API citées ci-dessus ne sera nécessaire. Le matériel et les logiciels nécessaires au développement sont fournis par Pharos Consulting.

Comme dit ci-dessus, un collaborateur de Pharos Consulting sera dédié pour faire le lien entre l'équipe de développement et OC-Pizza. Il devra donc être l'unique interlocuteur pour tout ce qui est avancée, du projet, fonctionnalités...

7.4 - Packaging/Livraison

L'application fera l'objet d'un déploiement sur le serveur de production préalablement préparé par OC_PIZZA.

Un dossier d'exploitation sera remis à cette occasion ainsi qu'un PV de livraison.

Les identifiants et mots de passe seront alors remis au client avec à sa charge le devoir de les modifier.

8 - GLOSSAIRE

Framework	Ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel
UML	Langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système
GIT	Logiciel de gestion de version décentralisé
Pip	Gestionnaire de paquets Python (Gère les inter-dépendances)