ACME

OC Pizza

Documentez votre système de gestion de pizzeria

Dossier de conception technique

Version 1

Auteur
Dylan Vallée
Analyste-programmeur

Table des matières

ACME

1 - VERSIONS

Auteur	Date	Description	Version
Dylan Vallée	14/01/2022	Création du document	1

2 - Introduction

2.1 - Objet du document

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

2.2 - Références

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

1. **DCF – OC Pizza** : Dossier de conception fonctionnelle de l'application
2. **DE – OC Pizza** : Dossier d'exploitation de l'application

ACME

2.3 - Références

2.3.1 - Contexte

« OC Pizza » est un jeune groupe de pizzeria en plein essor. Créé par Franck et Lola, le groupe est spécialisé dans les pizzas livrées ou à emporter. Il compte déjà 5 points de vente et prévoit d'en ouvrir au moins 3 de plus d'ici 6 mois.

Le système informatique actuel ne correspond plus aux besoins du groupe car il ne permet pas une gestion centralisée de toutes les pizzerias.
De plus, il est très difficile pour les responsables de suivre ce qui se passe dans les points de ventes.

2.3.2 - Enjeux et Objectifs

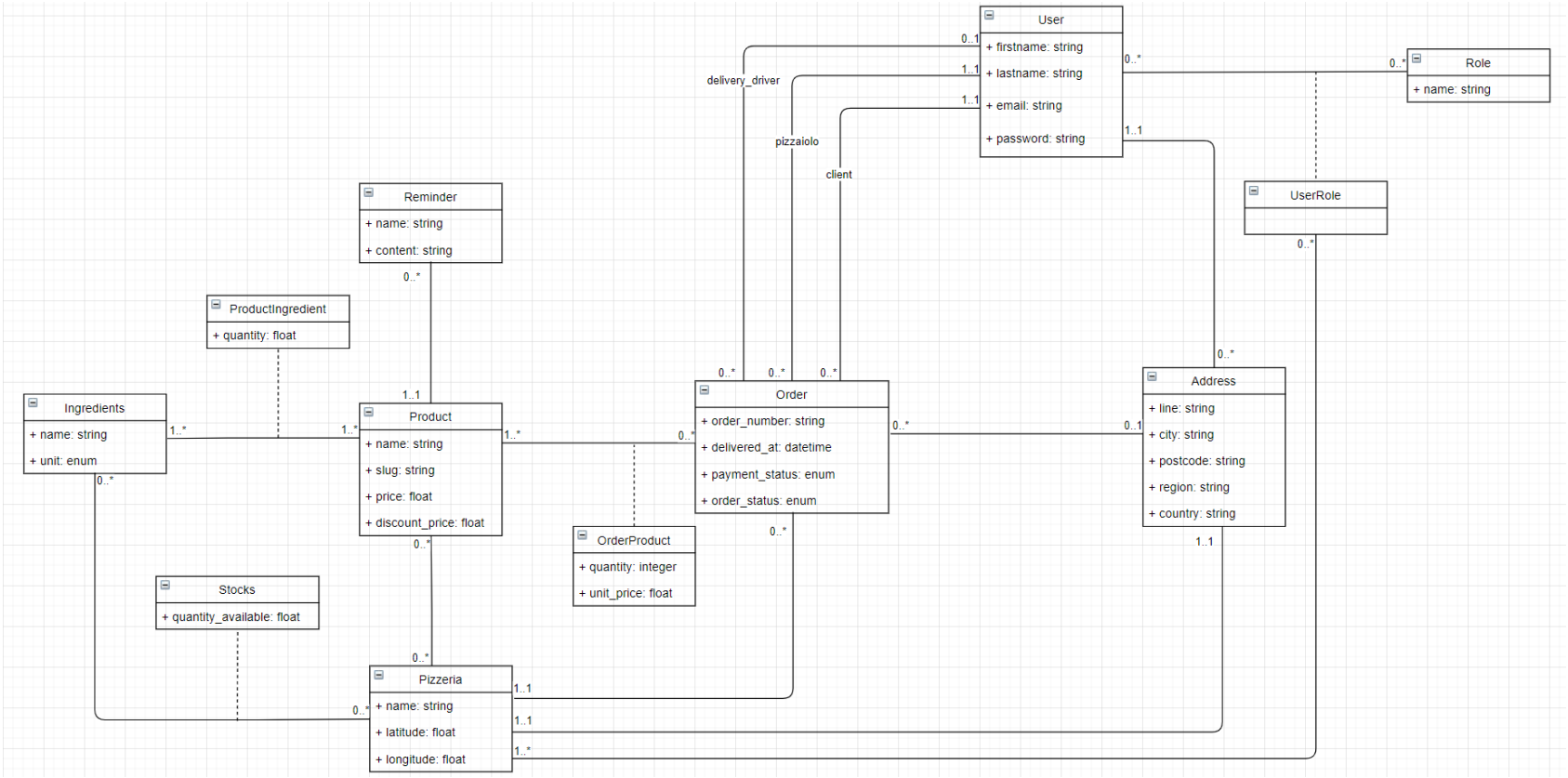
Ci-dessous les objectifs auxquels devra répondre le nouveau système afin de répondre aux besoins du client :

- Être plus efficace dans la gestion des commandes, de leur réception à leur livraison en passant par leur préparation
- Suivre en temps réel les commandes passées, en préparation et en livraison
- Suivre en temps réel le stock d'ingrédients restants pour savoir quelles pizzas peuvent encore être réalisées
- Proposer un site Internet pour que les clients puissent :
    - Passer leurs commandes, en plus de la prise de commande par téléphone ou sur place
    - Payer en ligne leur commande s'ils le souhaitent – sinon, ils paieront directement à la livraison
    - Modifier ou annuler leur commande tant que celle-ci n'a pas été préparée.
- Proposer un aide-mémoire aux pizzaiolos indiquant la recette de chaque pizza

Le système doit être livré lors de l'ouverture des 3 nouvelles pizzerias, c'est-à-dire dans six mois.

3 - Architecture Technique

3.1 - Diagramme de classe



ACME

3.2 - Présentation des classes

3.2.1 - Classe « User »

Propriété	**Type**
firstname	string
lastname	string
email	string
password	string

3.2.2 - Classe « Role »

Propriété	**Type**
name	string

3.2.3 - Classe « Address »

Propriété	**Type**
line	string
city	string
postcode	string
region	string
country	string

3.2.4 - Classe « Pizzeria »

Propriété	**Type**
name	string
latitude	float
longitude	float

3.2.5 - Classe « Ingredients »

Propriété	**Type**
name	string
quantity	integer
unit	enumeration

ACME

3.2.6 - Classe « Product »

Propriété	Type
name	string
slug	string
price	float
discount_price	float

3.2.7 - Classe « Reminder »

Propriété	Type
name	string
content	string

3.2.8 - Classe « Order »

Propriété	Type
order_number	string
delivered_at	string
payment_status	enumeration
order_status	enumeration

3.2.9 - Classe « UserRole »

Propriété	Type
user	User
role	Role

3.2.10 - Classe « Stocks »

Propriété	Type
quantity_available	float

3.2.11 - Classe « ProductIngredient »

Propriété	Type
quantity	float

ACME

3.2.12 - Classe « OrderProduct »

Propriété	Type
quantity	integer
unit_price	float

3.2.13 - Classe « Unit »

Propriété	Type
name	string
symbol	string

3.2.14 - Classe « PaymentStatus »

Propriété	Type
name	string
slug	string

3.2.15 - Classe « OrderStatus »

Propriété	Type
name	string
slug	string

ACME

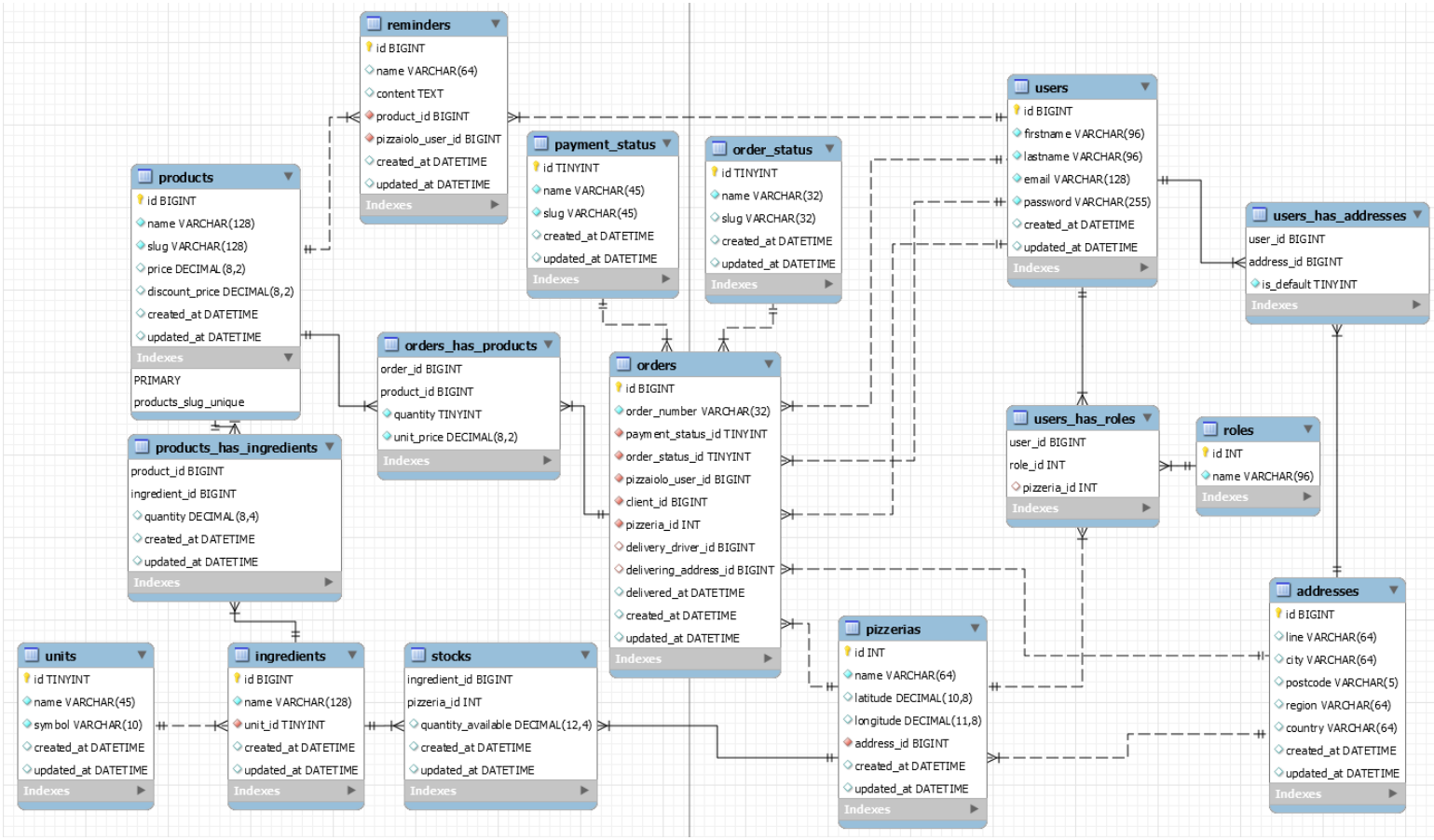
3.3 - Cardinalités des relations

Relation	**Cardinalité**
User – Role	0, n (table d'association « user_has_roles ») Un utilisateur peut avoir zéro ou plusieurs rôles. Un rôle peut être attribué à aucun ou plusieurs utilisateurs.
User – Pizzeria	0, n (table d'association « user_has_roles ») Un utilisateur (pizzaiolo, administrateur, livreur) peut appartenir à zéro ou plusieurs pizzerias. Une pizzeria peut contenir aucun ou plusieurs utilisateurs (pizzaiolo, administrateur, livreur).
User – Address	0, n / 1, 1 inversé Un utilisateur peut avoir zéro ou plusieurs adresses. Une adresse appartient à un utilisateur.
User – Order (client)	0, n / 1, 1 inversé Un client peut avoir zéro ou plusieurs commandes. Une commande concerne un seul client.
User – Order (pizzaiolo)	0, n / 1, 1 inversé Un pizzaiolo peut avoir préparé zéro ou plusieurs commandes. Une commande est préparée par un seul pizzaiolo.
User – Order (delivery_driver)	0, n / 0, 1 inversé Un livreur peut avoir transporté zéro ou plusieurs commandes. Une commande peut être prise en charge par un livreur.

ACME

Pizzeria – Address	1, 1 / 1, 1 inversé Une pizzeria possède une seule adresse. Une adresse concerne une seule pizzeria.
Pizzeria – Order	0, n / 1, 1 inversé Une pizzeria possède zéro ou plusieurs commandes. Une commande est passée chez une seule pizzeria.
Pizzeria – Ingredient	0, n (table d'association « stocks ») Une pizzeria ne possède aucun ingrédient ou en a plusieurs. Un ingrédient peut être stocké chez aucune ou plusieurs pizzerias.
Product – Order	0, n / 1, n inversé Un produit peut avoir été commandé zéro ou plusieurs fois. Une commande possède au minimum un produit.
Product – Reminder	0, n / 1, 1 inversé Un produit peut avoir aucun ou plusieurs pense-bêtes. Un pense-bête concerne un seul produit à la fois.
Product – Ingredient	1, n / 1, n Un produit possède au minimum un ingrédient. Un ingrédient peut être dans la composition d'au moins un produit.
Order – Address	0, 1 / 0, n inversé Une commande peut n'avoir aucune ou une seule adresse de livraison. Une adresse peut être utilisée par aucune ou plusieurs commandes pour une livraison.

3.4 - Modèle physique de données



ACME

3.4.1 - Table « users »

Colonne	Type / Index
id	BIGINT / AUTO INCREMENT / PRIMARY KEY
firstname	VARCHAR(96) / NOT NULL
lastname	VARCHAR(96) / NOT NULL
email	VARCHAR(128) / NOT NULL UNIQUE “users_email_unique”
password	VARCHAR(255) / NOT NULL
created_at	DATETIME
updated_at	DATETIME

3.4.2 - Table « roles »

Colonne	Type / Index
id	INT / AUTO INCREMENT / PRIMARY KEY
name	VARCHAR(96) NOT NULL
created_at	DATETIME
updated_at	DATETIME

3.4.3 - Table « addresses »

Colonne	Type / Index
id	BIGINT / AUTO INCREMENT / PRIMARY KEY
line	VARCHAR(64)
city	VARCHAR(64)
postcode	VARCHAR(5)
region	VARCHAR(64)
country	VARCHAR(64)
created_at	DATETIME
updated_at	DATETIME

ACME

3.4.4 - Table « pizzerias »

Colonne	Type / Index
id	INT / AUTO INCREMENT / PRIMARY KEY
name	VARCHAR(64) / NOT NULL
longitude	DECIMAL(10, 8)
latitude	DECIMAL(11, 8)
address_id	BIGINT / NOT NULL FOREIGN KEY “fk_pizzerias_addresses”
created_at	DATETIME
updated_at	DATETIME

3.4.5 - Table « units »

Colonne	Type / Index
id	TINYINT / AUTO INCREMENT / PRIMARY KEY
name	VARCHAR(45) / NOT NULL
symbol	VARCHAR(10) / NOT NULL
created_at	DATETIME
updated_at	DATETIME

3.4.6 - Table « ingredients »

Colonne	Type / Index
id	BIGINT / AUTO INCREMENT / PRIMARY KEY
name	VARCHAR(128) / NOT NULL
unit_id	TINYINT / NOT NULL FOREIGN KEY “fk_ingredients_units”
created_at	DATETIME
updated_at	DATETIME

ACME

3.4.7 - Table « products »

Colonne	Type / Index
id	BIGINT / AUTO INCREMENT / PRIMARY KEY
name	VARCHAR(45) / NOT NULL
slug	VARCHAR(45) / NOT NULL UNIQUE “products_slug_unique”
price	DECIMAL(8, 2)
discount_price	DECIMAL(8, 2)
created_at	DATETIME
updated_at	DATETIME

3.4.8 - Table « reminders »

Colonne	Type / Index
id	BIGINT / AUTO INCREMENT / PRIMARY KEY
name	VARCHAR(64)
content	TEXT
product_id	BIGINT / NOT NULL FOREIGN KEY “fk_reminders_products”
pizzaiolo_user_id	BIGINT / NOT NULL FOREIGN KEY “fk_reminders_users_pizzaiolo”
created_at	DATETIME
updated_at	DATETIME

3.4.9 - Table « order_status »

Colonne	Type / Index
id	TINYINT / AUTO INCREMENT / PRIMARY KEY
name	VARCHAR(32) / NOT NULL
slug	VARCHAR(32) / NOT NULL UNIQUE “order_status_slug_unique”
created_at	DATETIME
updated_at	DATETIME

ACME

3.4.10 - Table « payment_status »

Colonne	Type / Index
id	TINYINT / AUTO INCREMENT / PRIMARY KEY
name	VARCHAR(45) / NOT NULL
slug	VARCHAR(45) / NOT NULL UNIQUE “payment_status_slug_unique”
created_at	DATETIME
updated_at	DATETIME

3.4.11 - Table « orders »

Colonne	Type / Index
id	BIGINT / AUTO INCREMENT / PRIMARY KEY
order_number	VARCHAR(32) / NOT NULL UNIQUE “orders_order_number_unique”
payment_status_id	TINYINT / NOT NULL FOREIGN KEY “fk_orders_payment_status”
order_status_id	TINYINT / NOT NULL FOREIGN KEY “fk_orders_order_status”
pizzaiolo_user_id	BIGINT / NOT NULL FOREIGN KEY “fk_orders_users_pizzaiolo”
client_id	BIGINT / NOT NULL FOREIGN KEY “fk_orders_users_client”
pizzeria_id	INT / NOT NULL FOREIGN KEY “fk_orders_pizzerias”
delivery_driver_id	BIGINT FOREIGN KEY “fk_orders_users_delivery_driver”
delivering_address_id	BIGINT FOREIGN KEY “fk_orders_addresses”
created_at	DATETIME
updated_at	DATETIME

ACME

3.4.12 - Table d'association « users_has_roles »

Colonne	Type / Index
user_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_users_has_roles_users"
role_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_users_has_roles_roles"
pizzeria_id	INT / NOT NULL FOREIGN KEY "fk_users_has_roles_pizzerias"
created_at	DATETIME
updated_at	DATETIME

3.4.13 - Table d'association « users_has_addresses »

Colonne	Type / Index
user_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_users_has_addresses_users"
address_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_users_has_addresses_addresses"
is_default	TINYINT(1) DEFAULT 0 / NOT NULL
created_at	DATETIME
updated_at	DATETIME

3.4.14 - Table d'association « stocks »

Colonne	Type / Index
ingredient_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_stocks_ingredients"
pizzeria_id	INT / PRIMARY KEY FOREIGN KEY "fk_stocks_pizzerias"
quantity_available	DECIMAL(12,4)
created_at	DATETIME
updated_at	DATETIME

3.4.15 - Table d'association « products_has_ingredients »

Colonne	Type / Index
product_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_product_ingredients_products"
ingredient_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_product_ingredients_ingredients"
quantity	DECIMAL(8,4)
created_at	DATETIME
updated_at	DATETIME

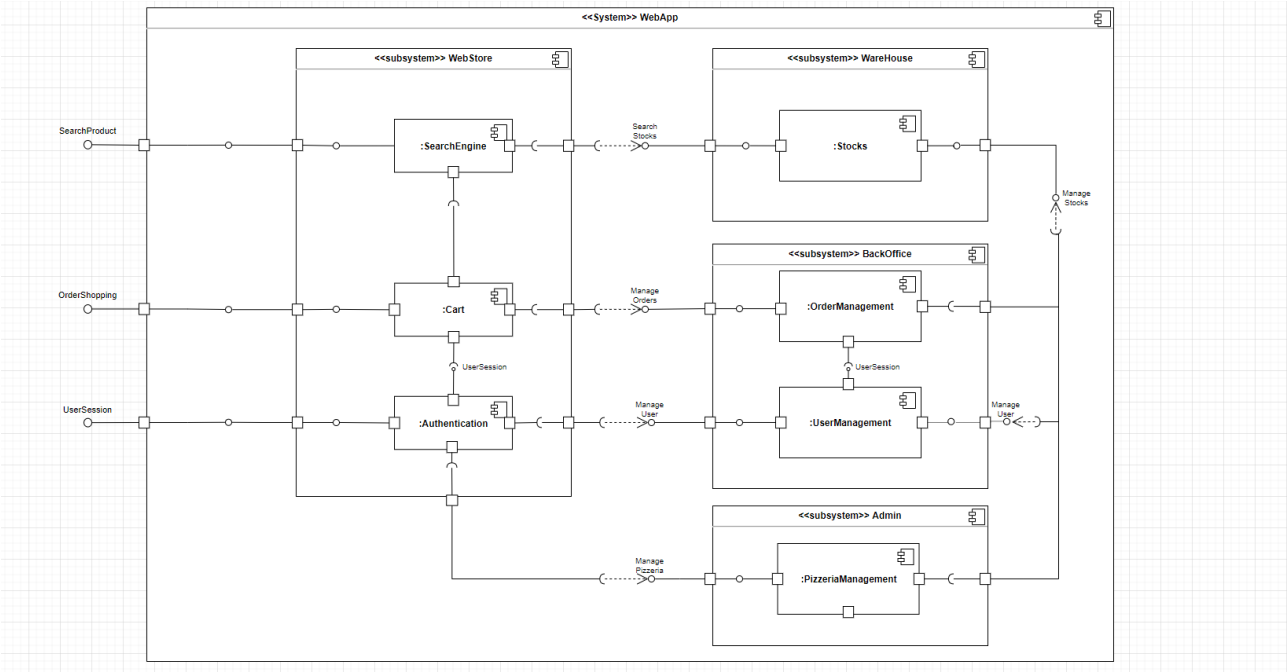
3.4.16 - Table d'association « orders_has_products »

Colonne	Type / Index
order_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_orders_products_orders"
product_id	BIGINT / PRIMARY KEY FOREIGN KEY "fk_orders_products_products"
quantity	TINYINT / NOT NULL
unit_price	DECIMAL(8,2) / NOT NULL
created_at	DATETIME
updated_at	DATETIME

3.5 - Application Web

La pile logicielle est la suivante :

- Application **Python 3** (Framework Django ^3.2.5)
- Serveur d'application **Gunicorn 20.1.0** / **Nginx 1.21.6**
- SGBDR **MySQL** 8.0.28



ACME

3.5.1 - Sous-système « Webstore »

3.5.1.1 - SearchEngine

Permet la recherche de produit dans le catalogue ou bien dans les stocks.

3.5.1.2 - Composant « Cart »

Permet le stockage de produit(s) afin de générer une commande à partir de ceux-là.

3.5.1.3 - Composant « Authentification »

Permet l'accès à des fonctionnalités indisponible lorsqu'un utilisateur est en mode « guest ». Acquisition de droits/permissions en fonction du rôle de l'utilisateur.

3.5.2 - Sous-système « Warehouse »

3.5.2.1 - Composant « Stocks »

Permet la manipulation des ressources « Product » et « Ingredients » (création, modification, suppression).

3.5.3 - Sous-système « BackOffice »

3.5.3.1 - Composant « OrderManagement »

Gestion de la ressource « Order », càd création, mise à jour, livraison, paiement, etc… ainsi que la lecture des commandes passées.

3.5.3.2 - Composant « UserManagement »

Gestion de la ressource « User » (création, modification et suppression).

3.5.4 - Sous-système « Admin »

3.5.4.1 - Composant « PizzeriaManagement »

Gestion des composants ci-dessous :

- WareHouse.Stocks
- BackOffice.UserManagement
- BackOffice.OrderManagement

4 - Architecture de Deploiement

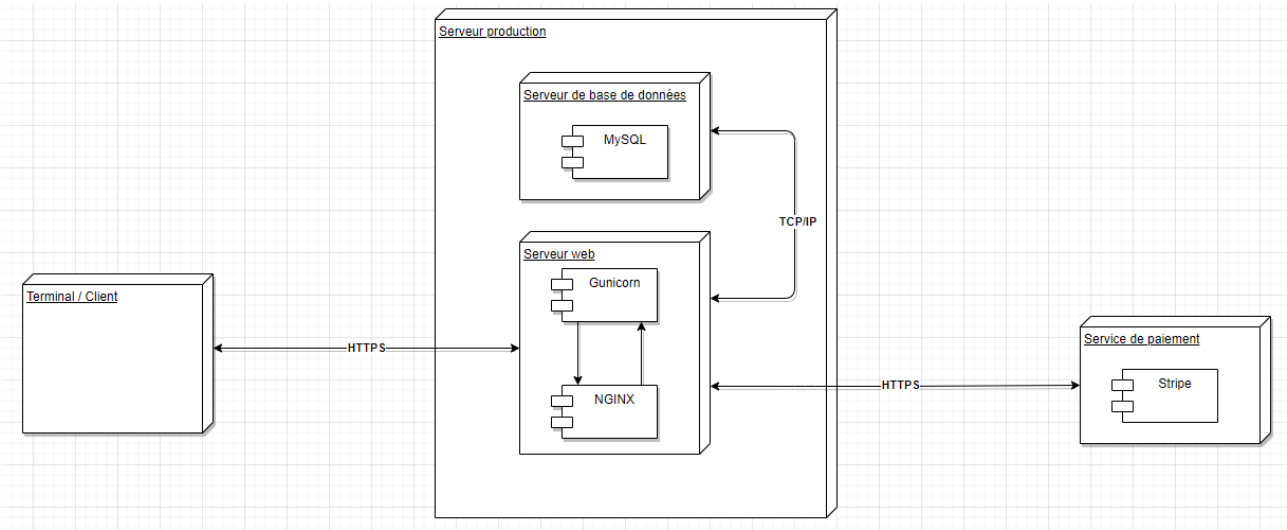


Diagramme UML de déploiement

4.1 - Serveur de Base de données

Le serveur de base de données est hébergé sur le serveur de production.

Caractéristiques techniques :

- OS Linux Debian 9 (Stretch)
- PostgreSQL 14.2

Le serveur de base de données peut être sur un autre serveur que celui où se situe l'application.

4.2 - Serveur Web

Le serveur web permet de servir l'application sur internet.

Caractéristiques techniques :

- Serveur web NGINX
- Serveur web WSGI Gunicorn 20.1.0

4.3 - Service de paiement

Stripe est l'infrastructure de paiement utilisée afin de procéder à des transactions bancaires.

Caractéristiques techniques :

- Stripe version 2020-08-27

ACME

5 - ARCHITECTURE LOGICIELLE

5.1 - Principes généraux

Les sources et versions du projet sont gérées par **Git**, les dépendances et le packaging par **PIP** (Package installer for Python). Le dépôt distant est hébergé sur Github.

5.1.1 - Les couches

L'architecture applicative est la suivante :

- une couche **business** : responsable de la logique métier du composant
- une couche **model** : implémentation du modèle des objets métiers
- une couche **template** : implémentation de l'interface utilisateur

5.1.2 - Les modules

L'application est divisée en plusieurs modules (applications dans le cas de Django).

5.1.2.1 - Module « ocpizza »

C'est le module responsable de la configuration entière du projet (paramètres de l'application, serveur WSGI/ASGI, etc...)

5.1.2.2 - Module « account »

Ce module est responsable de l'authentification, l'enregistrement et la gestion de l'utilisateur sur le site.

5.1.2.3 - Module « home »

Le module « Home » fournit les principales pages statiques du site web (page d'accueil, conditions générales, etc...)

5.1.2.4 - Module « product »

Ce module ajoute le « modèle » représentant les produits en base de données. Il s'occupe également de l'affichage en liste des produits et est faiblement couplé au module « substitute »

5.1.2.5 - Module « order »

Module intrinsèquement lié au module « product » car il est dépendant de celui-ci, il permet de créer une commande contenant un ou plusieurs produits.

ACME

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 de Django :

```
oc-pizza
├─ ocpizza
│  ├─ account
│  │  ├─ migrations
│  │  ├─ static
│  │  ├─ templates
│  │  ├─ tests
│  │  ├─ __init__.py
│  │  ├─ admin.py
│  │  ├─ apps.py
│  │  ├─ forms.py
│  │  ├─ models.py
│  │  ├─ urls.py
│  │  └─ views.py
│  ├─ home
│  ├─ product
│  │  └─ management
│  │     └─ commands
│  │        └─ populate.py
│  ├─ ocpizza
│  │  ├─ .env
│  │  ├─ .env.example
│  │  ├─ asgi.py
│  │  ├─ settings.py
│  │  ├─ utils.py
│  │  └─ wsgi.py
│  ├─ order
│  ├─ templates
│  │  └─ registration
│  ├─ manage.py
│  └─ setup.cfg
├─ .gitignore
├─ .travis.yml
├─ Procfile
├─ README.md
├─ release-tasks.sh
└─ requirements.txt
```

ACME

6 - POINTS PARTICULIERS

6.1 - Gestion des logs

La gestion des logs est entièrement prise en charge par la plateforme « Sentry » à laquelle l'application envoie des données régulièrement par le biais du package Python « sentry-sdk ».

6.2 - Fichiers de configuration

6.2.1 - Application web

Le fichier de configuration globale de l'application est le suivant :

6.2.1.1 - Fichier .env

Le fichier d'environnement permet de définir des variables d'environnement pour l'application. Il est accessible au chemin suivant :

oc-pizza/ocpizza/ocpizza/.env

6.3 - Ressources

Les ressources sont stockées dans un dossier « static » présent dans chaque application django. Pour que ces ressources soient regroupées utilisées en production, il faut utiliser la commande suivante :

python manage.py collectstatic

6.4 - Environnement de développement

L'environnement de développement peut être changé via le fichier d'environnement qui spécifie une variable d'environnement que l'application utilise afin de savoir dans quel environnement elle se trouve. Le nom de la variable est « APP_ENV »

ACME

7 - GLOSSAIRE

WSGI	Web Server Gateway Interface