

OC PIZZA

Mise en place d'un système informatique

Dossier de conception technique

Version OC_P6[01]

Auteur

Gabrielle Azadian

Développeur d'application junior

Table des matières

1_VERSIONS.....	4
2_INTRODUCTION.....	5
2.1_Objet du document.....	5
3_LE DOMAINE FONCTIONNEL.....	6
3.1_Référentiel Diagramme de Classe UML	6
3.1.1_Règles de gestion.....	7
4_ARCHITECTURE TECHNIQUE.....	8
4.1_Application Web	8
4.1.1_Composants	8
4.1.2_Composants	9
4.1.3_Composants	10
4.1.4_Composants	11
5_ARCHITECTURE DE DÉPLOIEMENT	12
5.1_Serveur de Base de Données PostgreSQL	12
5.2_Serveur d'application Gunicorn	12
5.3_Serveur Web NGINX	13
7_GLOSSAIRE	13

1_VERSIONS

Auteur	Date	Descriptions	Versions
GA	24/03/2021	Version OC_V01 Dossier technique	OC_P6[01]

2_INTRODUCTION

2.1_Ojet du document

Le présent document constitue le dossier de conception technique d'un site responsive OC Pizza.

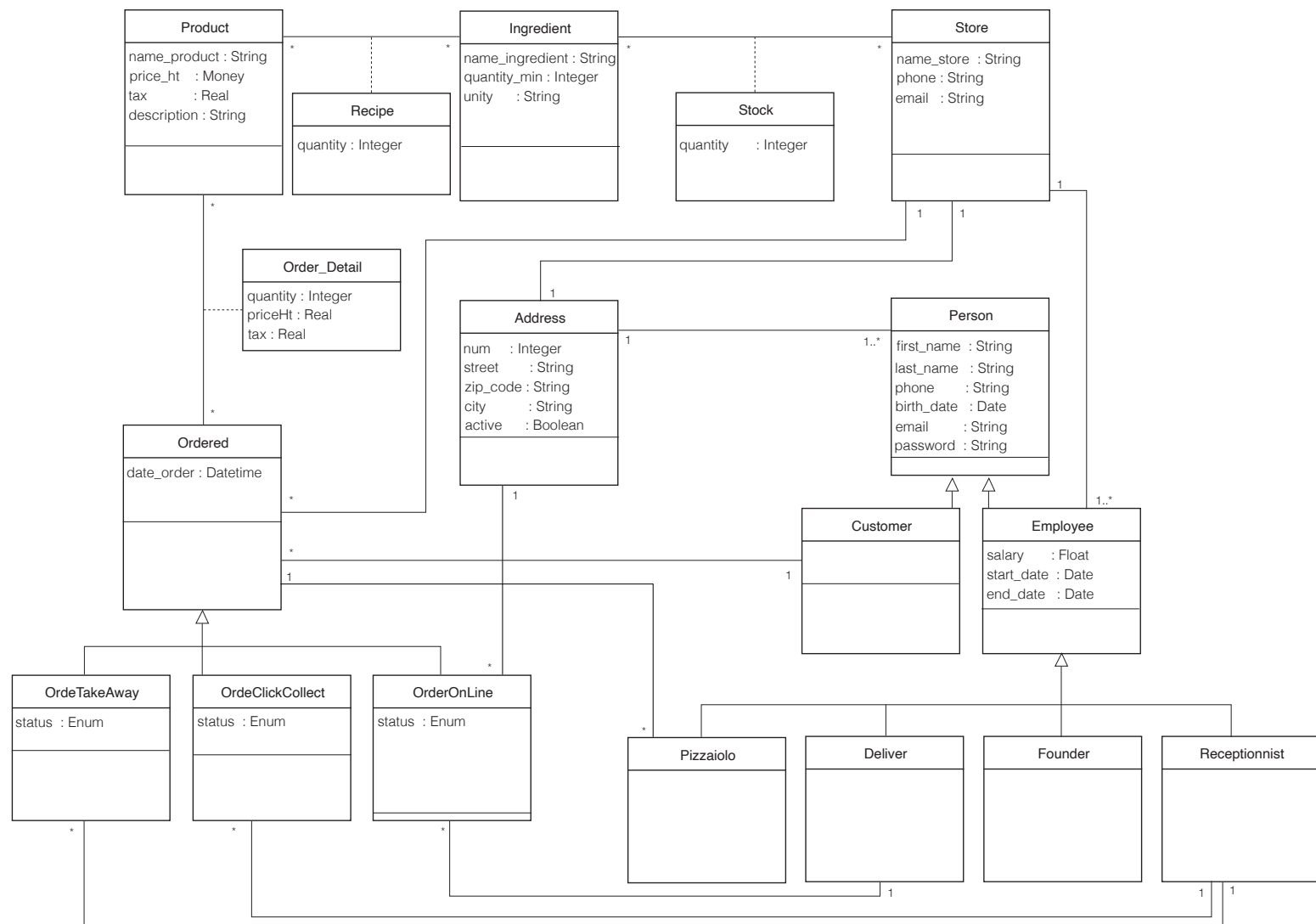
Objectif du document : développer un logiciel sur mesure pour améliorer la gestion des restaurants OC Pizza

Les éléments du présents dossiers découlent :

- De l'architecture technique : diagramme de composants
- De l'architecture de Déploiement : serveur de Base de données

3_LE DOMAINE FONCTIONNEL

3.1_Référentiel Diagramme de Classe UML



Le diagramme de Classe respecte **la première forme normale**. En effet, la classe Address est décomposée (num, street ...) et possède une dépendance fonctionnelle avec les Classes Person et OrderOnLine.

Il y a **3 tables d'associations** :

- Une table d'association entre **Product** et **Ingredients** qui représente les recettes (l'aide-mémoire pour le pizzaiolo)
- Entre **Ingredients** et **Store** qui représente les stocks (visualiser les stocks en temps réel)
- Entre **Product** et **Order** qui représente le détail de la commande (quantité, prix, tva). Cette table me permet de garder une trace du prix si il vient à changer ou si le produit est supprimé.

Dans la classe **Address**, l'attribut active de **type boolean** me permet de garder l'adresse du client si il venait à déménager. Si le client change d'adresse, l'ancienne adresse est **False**.

J'ai opté pour l'héritage pour les classes **Person** et **Ordered**. J'aurai pour tout simplement au lieu de créer des sous tables (Pizzaiolo, ...) créer une table **Rôle** et créer un lien entre **Employee** et **Rôle**. Le fait de créer ces tables me permet de créer un lien avec les classes **OrderOnline** avec **Deliver**, **OrderTakeaway** avec **Recetionnist** et enfin **Order** avec **Pizzaiolo**, **Customer** (puisque les clients peuvent commander un produit de toutes sortes (take away, on line, click & collect)).

3.1.1_Règles de gestion

1. **Product --> Ingredient** : relation *many to many* entre ces classes génère une classe d'association **Recipe**. Le produit (Product) possède plusieurs ingrédients (farine, tomates, mozzarella ...).
2. **Ingredient --> Store** : relation *many to many* génère une classe d'association **Stock**. Les ingrédients sont stockés dans plusieurs restaurants. Les 5 points de vente possèdent des stocks d'ingrédients.
3. **Product --> Ordered** : relation *many to many* génère une classe d'association **OrderDetail** (détail de la commande). Les produits sont commandés dans plusieurs commandes (provenant de client), les commandes peuvent contenir plusieurs produits.
4. **Store --> Address** : relation *one to one*. Chaque point de vente possède une adresse, une adresse référence un point de vente.
5. **Store --> Ordered** : relation *one to many*. Un point de vente s'occupe de plusieurs commandes, mais une commande est référencée dans un point de vente en particulier.
6. **Customer, Employee héritent de la classe Person --> Address** : relation *one to many*. Une adresse référence une seule personne, mais une personne peut posséder plusieurs adresses.
7. **OrderOnline, OrderTakeAway, OrderClickCollect héritent de la classe Ordered --> Customer** : relation *many to one*. Un client peut commander plusieurs commandes, une commande est réalisée, commandée par un seul client.
8. **Pizzaiolo hérite de la classe Employee --> Ordered** : *many to one*. Un pizzaiolo peut s'occuper de plusieurs commandes (en ligne, click & collect ou en take away), une commande est référencés et réalisée par un pizzaiolo.
9. **Deliver hérite de la classe Employee --> OrderOnLine hérite de la classe Ordered** : *many to one*. Un livreur peut livrer plusieurs commandes, une commande est référencée et livrée par un livreur.
10. **Receptionnist hérite de la classe Employee --> OrderTakeAway, OrderClickCollect héritent de la classe Ordered** : *many to one*. Une réceptionniste peut s'occuper de plusieurs commandes, une commande est référencée et traitée par une réceptionniste.
11. **Address --> OrderOnLine hérite de la classe Ordered** : *one to many*. Une commande est livrée à une adresse en particulier, l'adresse possède plusieurs commandes à être livrer.

4_ARCHITECTURE TECHNIQUE

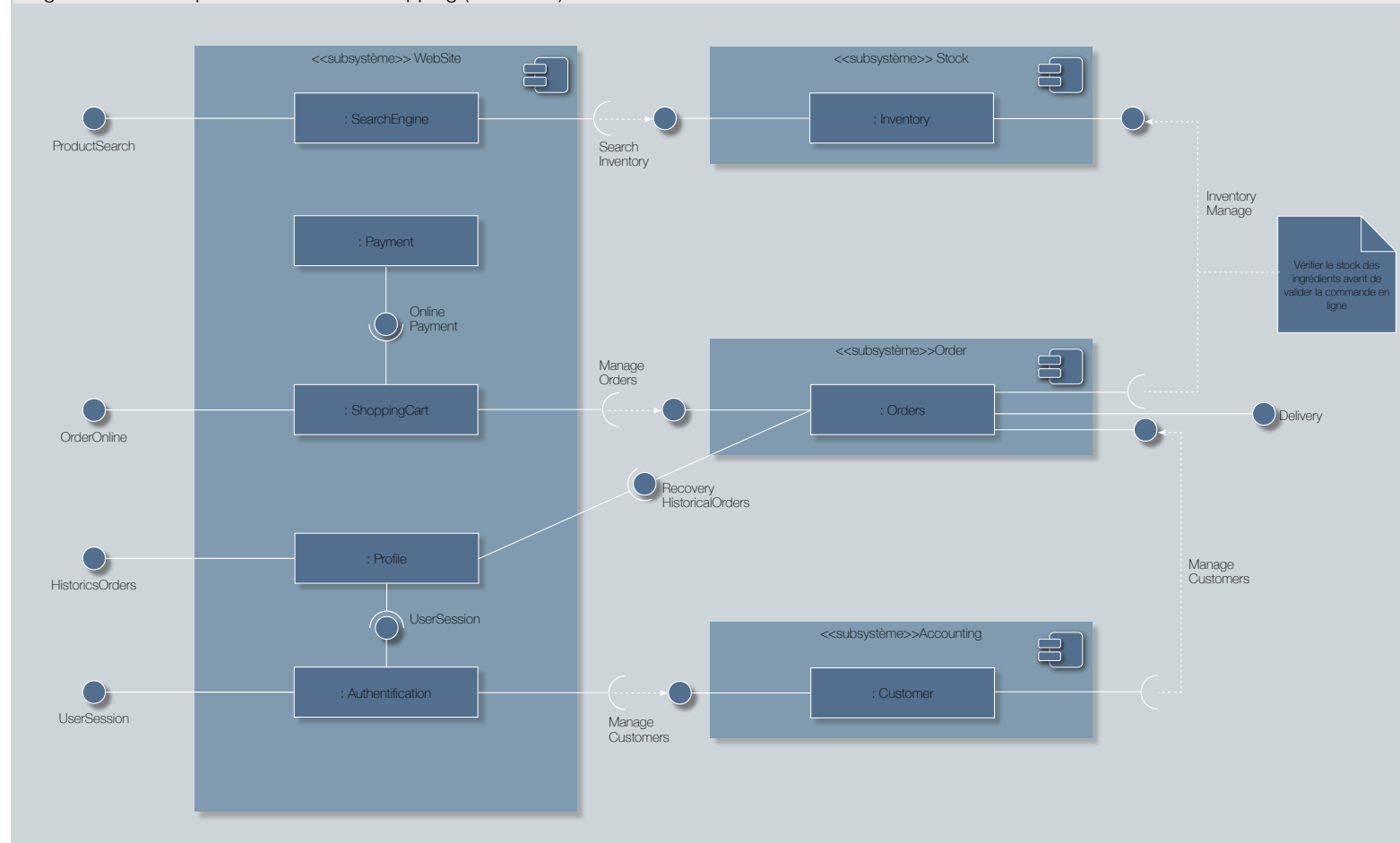
4.1_Application Web

La pile logicielle est la suivante :

- Application Python
- Serveur d'application **Gunicorn 20.0.4**
- Serveur Web **NGINX 1.22.0**
- Base de Données **PostgreSQL 13**

4.1.1_Composants

Diagramme de composants --> Online shopping (Customer)



Pour organiser la fonctionnalité **Online Shopping**, nous avons besoins du composant **:SearchEngine** (moteur de recherche).

Ce composant a besoin d'information requise du composant **:Inventory**. En effet, le composant fournis les informations concernant le produit recherché.

Le composant **:ShoppingCart** (Panier) a besoin d'une information fournie «*Lollipop*» qui est une commande en ligne dans ce cas.

Le composant **:Customer** fourni les informations requises pour le composant **:Authentication** (connexion au profil).

Si l'information est juste **UserSession**, le **:Profile** peut visualiser l'historique des commandes.

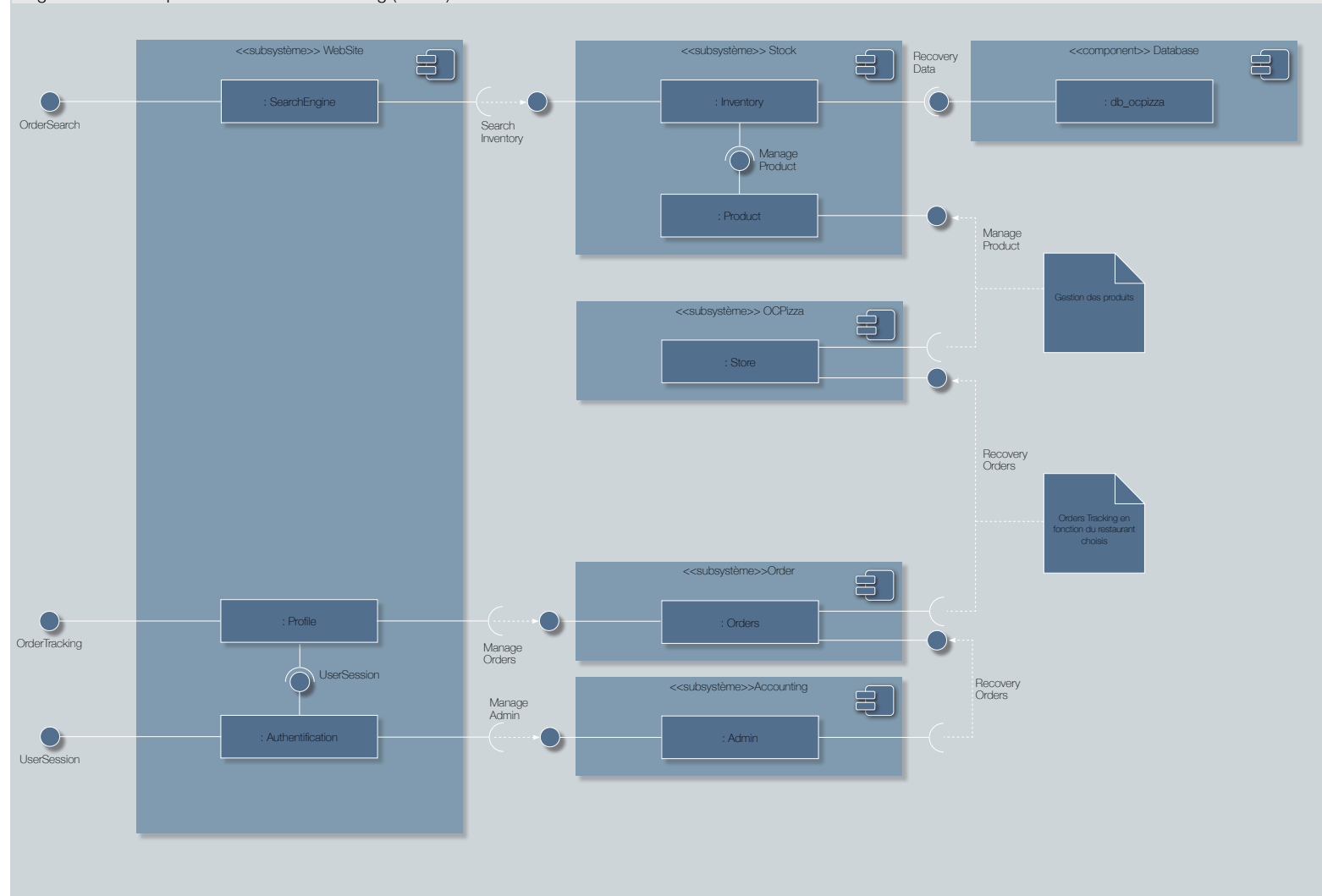
Pour valider la commande en ligne, le composant **:Orders** doit vérifier l'état des commandes **:Inventory**.

:Inventory fournies l'information (Interface fournie) au composant **:Orders** (interface requise).

Pour valider le panier, **:ShoppingCart** a besoin des informations bancaires (interface requise) pour effectuer un paiement en ligne **:Payment** (interface fournie).

4.1.2_Composants

Diagramme de composants --> Orders Tracking (Admin)

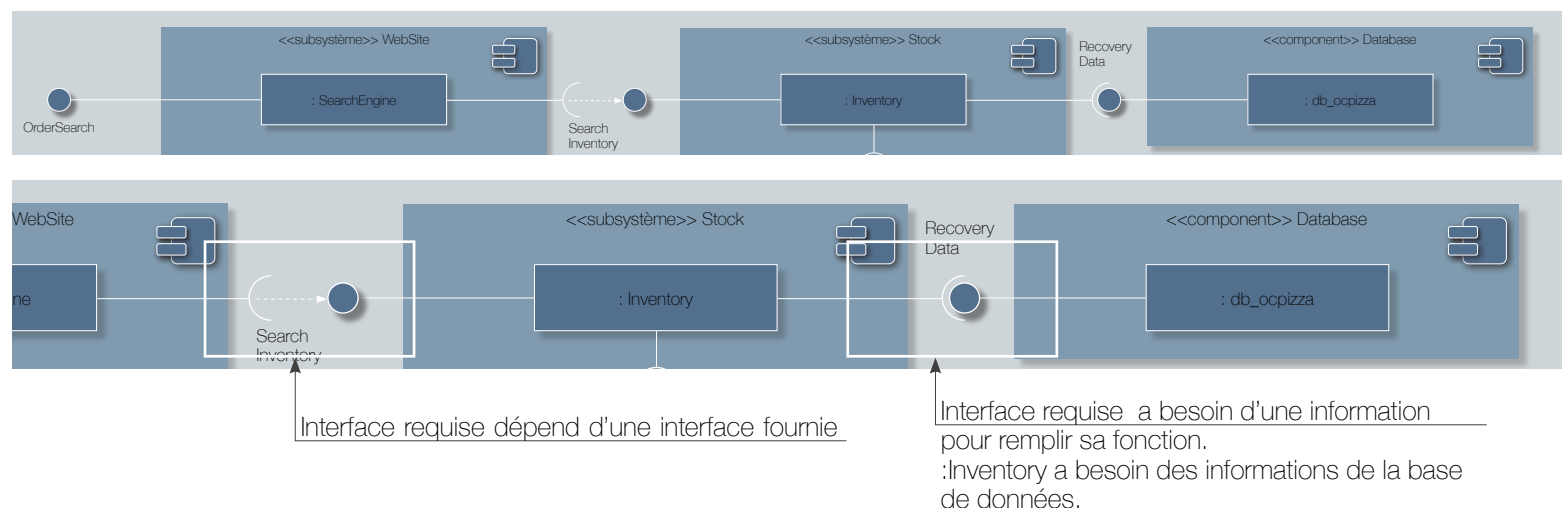


Pour organiser la fonctionnalité **Orders Tracking** par les fondateurs (Administrateur).

Le composant **:SearchEngine** (moteur de recherche) a besoin d'une interface fournie par une recherche **OrderSearch**. Ce composant pour remplir sa fonction a besoin (dépendance) d'une information (interface requise) fournie par le composant **:Inventory** qui lui se fourni dans la base de données **:db_ocpizza**.

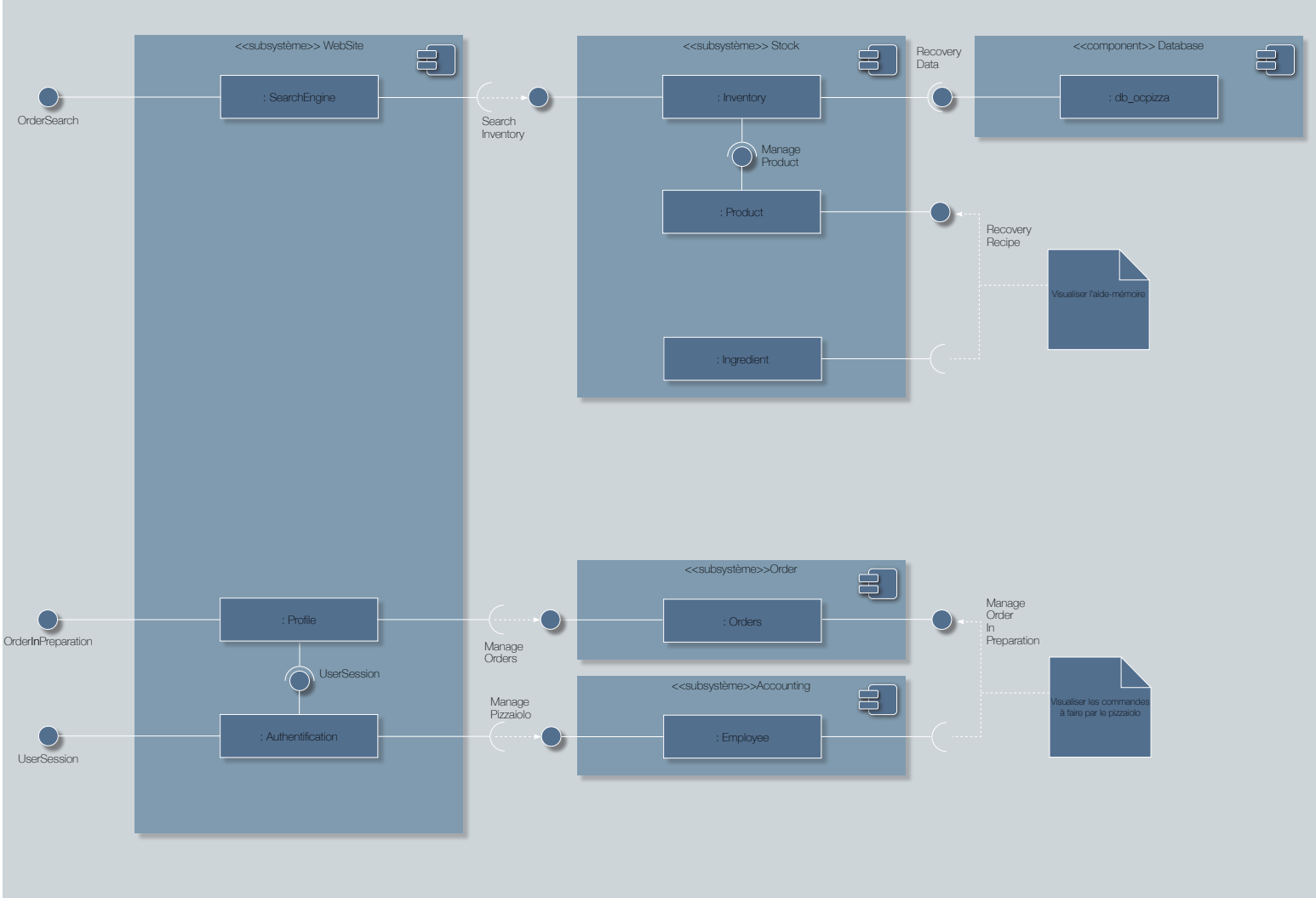
Pour pouvoir visualiser les états et suivre les commandes, les fondateurs doivent se connecter **:Profile** (interface fournie) **OrderTracking**. La consultation d'une commande **:Orders** dépend du composant **:Store** (interface fournie) **Recovery Orders**.

Pour gérer les produits, le composant **:Store** a besoin des informations fournies par le composant **:Product**



4.1.3_Composants

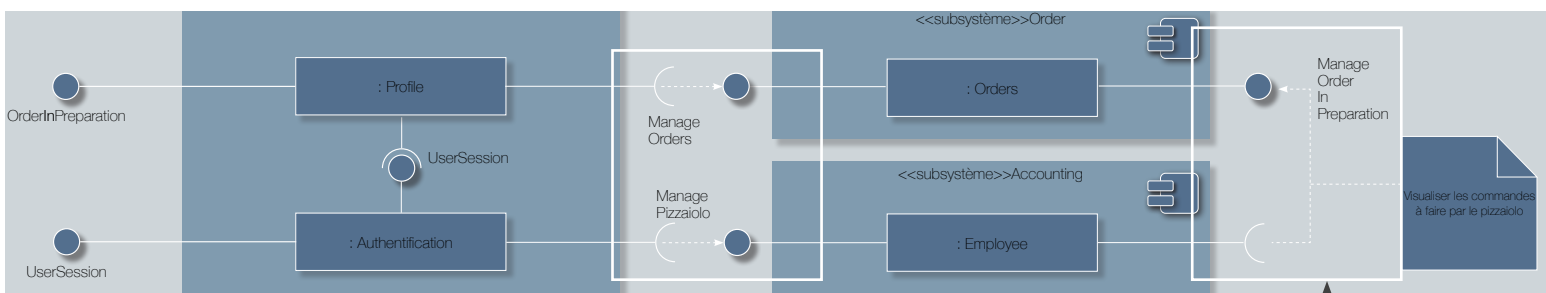
Diagramme de composants --> Order in preparation (Pizzaiolo)



Pour organiser la fonctionnalité **Order In preparation** permet aux Pizzaiolos de consulter les commandes à réaliser.

Pour obtenir la recette d'un produit, le composant **:Product** doit fournir les informations nécessaires au composant **:Ingredient**. C'est une relation de dépendance **Recovery Recipe**.

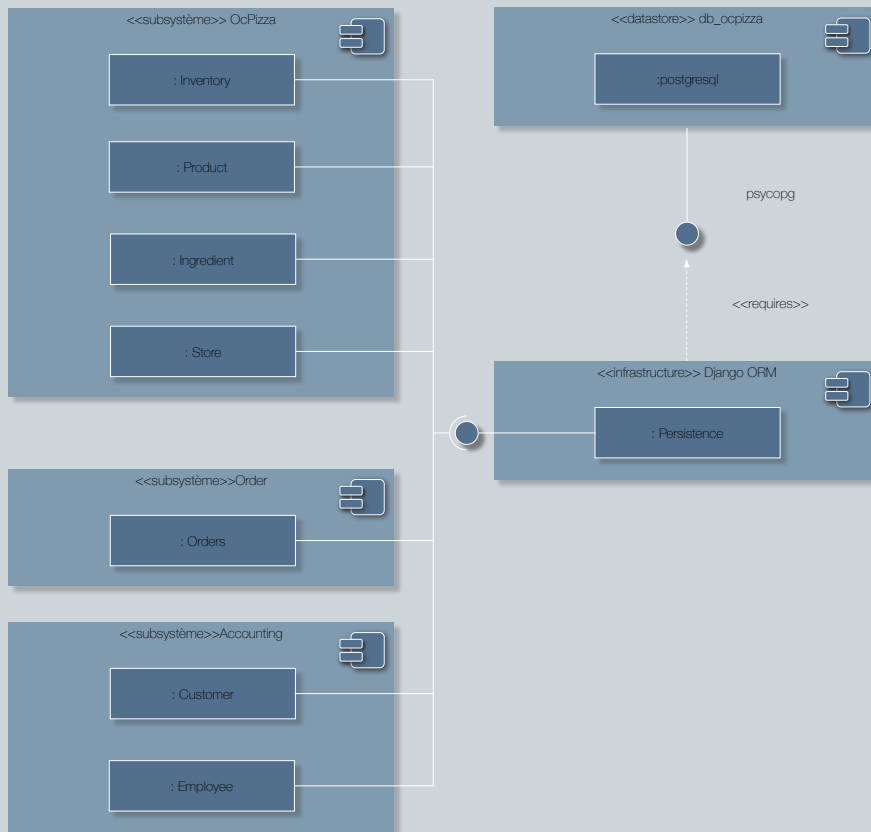
Le composant **:Employee** doit se connecter pour avoir accès aux commandes à réaliser. **:Employee** doit fournir ses informations au composant **:Authentication**.



Interface requise a besoin d'une information pour remplir sa fonction.
Relation de dépendance
:Employee a besoin des informations provenant du composant **:Orders** pour consulter les commandes à réaliser

4.1.4_Composants

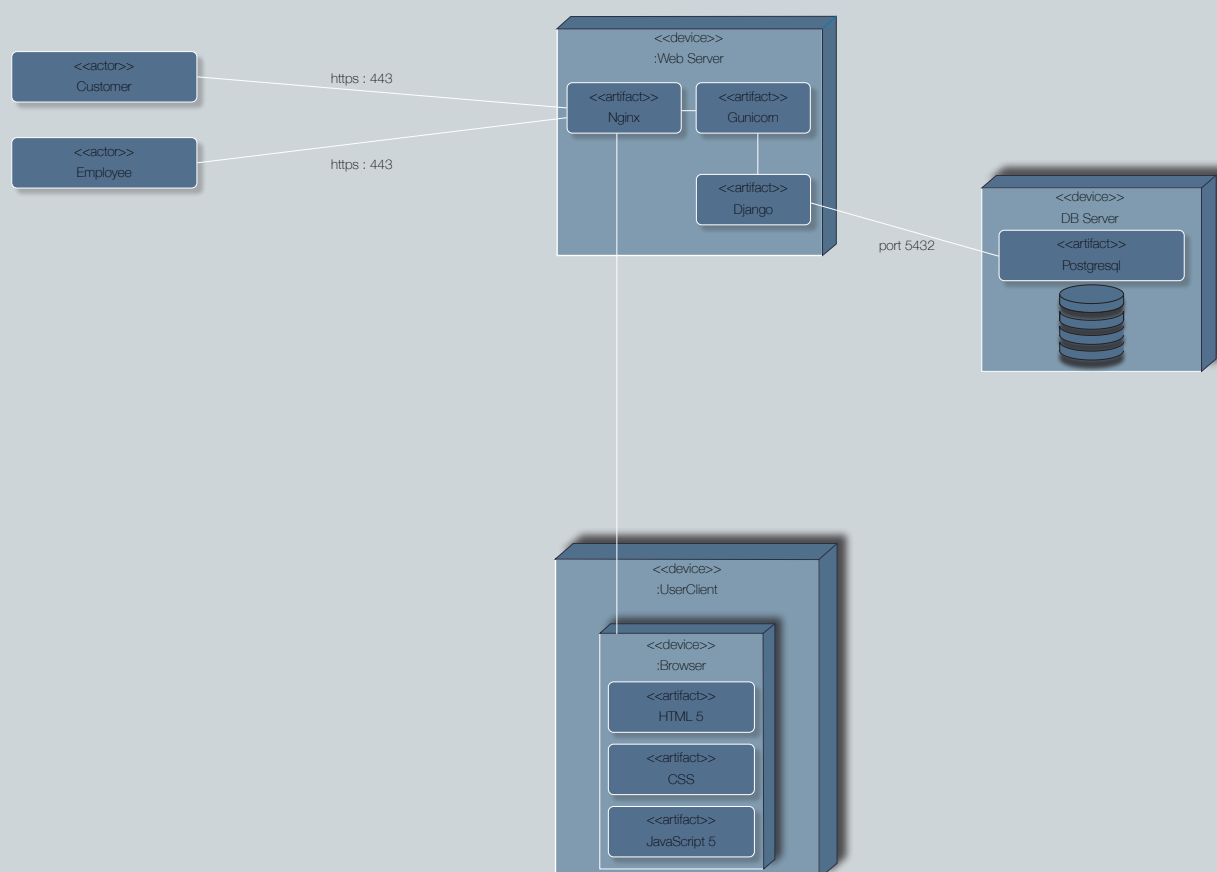
Diagramme de composants --> Database



Pour sauvegarder les objets dans la base de données.

5_ARCHITECTURE DE DÉPLOIEMENT

Diagramme de déploiement



Les clients et les employés se connectent sur le serveur HTTP **NGINX** et **GUNICORN** pour avoir accès aux pages réalisées en **HTML 5** et mis en forme par **CSS3**. Le framework **Django** se connecte via le port **5432** au SGBD **PostgreSQL**.

5.1_ Serveur de Base de Données PostgreSQL

La base de données est créée sur le système de gestion de base de données (SGBD) **PostgreSQL 13** installé sur un système **MacOs**. Le Modèle Physique de Données (ERD) a été réalisé sur **pgModeler V.0.9.2**. Pour insérer les données du fichier `insert.sql`, nous avons utilisé **pgAdmin p4 V 5.0**. Pour visualiser les résultats des requêtes du fichier `query.sql`, nous avons utilisé **Postico**.

5.2_ Serveur d'application Gunicorn



Nous allons utiliser le serveur d'application **Gunicorn V 20.0.4**.

Nous sommes obligés de faire appel à **Gunicorn** c'est un serveur **HTTP Python** pour **Unix** qui utilise les spécifications **WSGI (Web Server Gateway Interface)**.

5.3_ Serveur Web NGINX

Nous allons utiliser le serveur HTTP **ENGINE -X (NGINX)** en association avec le serveur **Gunicorn**.
NGINX n’interprète pas de code Python.

7 _GLOSSAIRE

Django	Un framework web qui contient un serveur web, un ORM, moteur de gabarit et un client test
Framework	«cadre de travail» en français, un framework offre un ensemble de module pour faciliter le travail d’un développeur, ainsi que lui permettre de commencer un projet rapidement.
Interfaces fournies 	Ce symbole représente les interfaces où un composant produit des informations utilisées par l’interface requise d’un autre composant.
Interfaces requises 	Ce symbole représente les interfaces où un composant a besoin d’informations pour remplir sa fonction.
pgModeler	Logiciel pour faire des base de données en Postgres.