OC PIZZA

Dossier de conception technique

Version 1.1

Auteur
AUBRUN Eric
Analalyste-programmeur

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
Eric AUBRUN	01/05/2021	Création du document	1.0
Eric AUBRUN	20/05/2021	Finalisation du document	1.1

2 - INTRODUCTION

2.1 - Objet du document

Le dossier de conception technique de l'application est rédigé à l'attention des développeurs, mainteneurs et de l'équipe technique d'OC PIZZA.

L'objectif de ce dossier est de détailler les contraintes spécifiques dont les développeurs vont devoir tenir compte pour coder l'application, ce document présente le langage ainsi que les conventions de développement, l'architecture logicielle et de déploiement de l'application.

Les éléments du présent dossier découlent des documents suivants :

- Mise en place d'un nouveau système informatique pour l'ensemble des pizzerias du groupe OC Pizza
- Concevez la solution technique d'un système de gestion de pizzeria

2.2 - Références

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

1. **PDOCPizza_02_fonctionnelle.pdf – 1.0** : Le Dossier de conception technique de l'application
2. **PDOCPizza_03_exploitation.pdf – 1.0** : Le Dossier d'exploitation de l'application
3. **PDOCPizza_04_livraison.pdf – 1.0 :** Le PV de livraison finale

3 - Architecture Technique

Nous proposons d'architecturer le produit en deux applications :

- Une application et son serveur traitant spécifiquement le front-end.
- Une seconde application installée sur un autre serveur, Angular pilotant la communication entre les deux applications à partir de requêtes HTTP.

3.1 - Application Web

La pile logicielle est la suivante :

- Pour la partie back-end :
    - PostgreSQL installé sur un serveur de données sous Linux, gérant l'authentification ainsi que le stock de produits et ingrédients, nécessaires à la confection des pizzas.
    - Le Framework Spring installé sur un serveur d'application tournant sous Linux, développé en Java, gérant plus particulièrement les différentes phases du cycle de vie d'une commande.
- Pour la partie front-end :
    - Le Framework Angular installé sur un serveur Web Apache, lui aussi sous Linux, accompagné d'un Framework CSS comme Bootstrap, permettant notamment d'obtenir un effet « Responsive Web Design », des librairies de composants Angular comme Ng-Bootstrap, en mesure par exemple de construire des fenêtres modales, des librairies générant des graphiques de type « camemberts » comme Chart.Js ou encore Lodash, une bibliothèque Javascript contenant des fonctions permettant par exemple de tester et manipuler des valeurs, créer des fonctions composites ou utiles pour des tableaux, objets ou chaînes de texte.

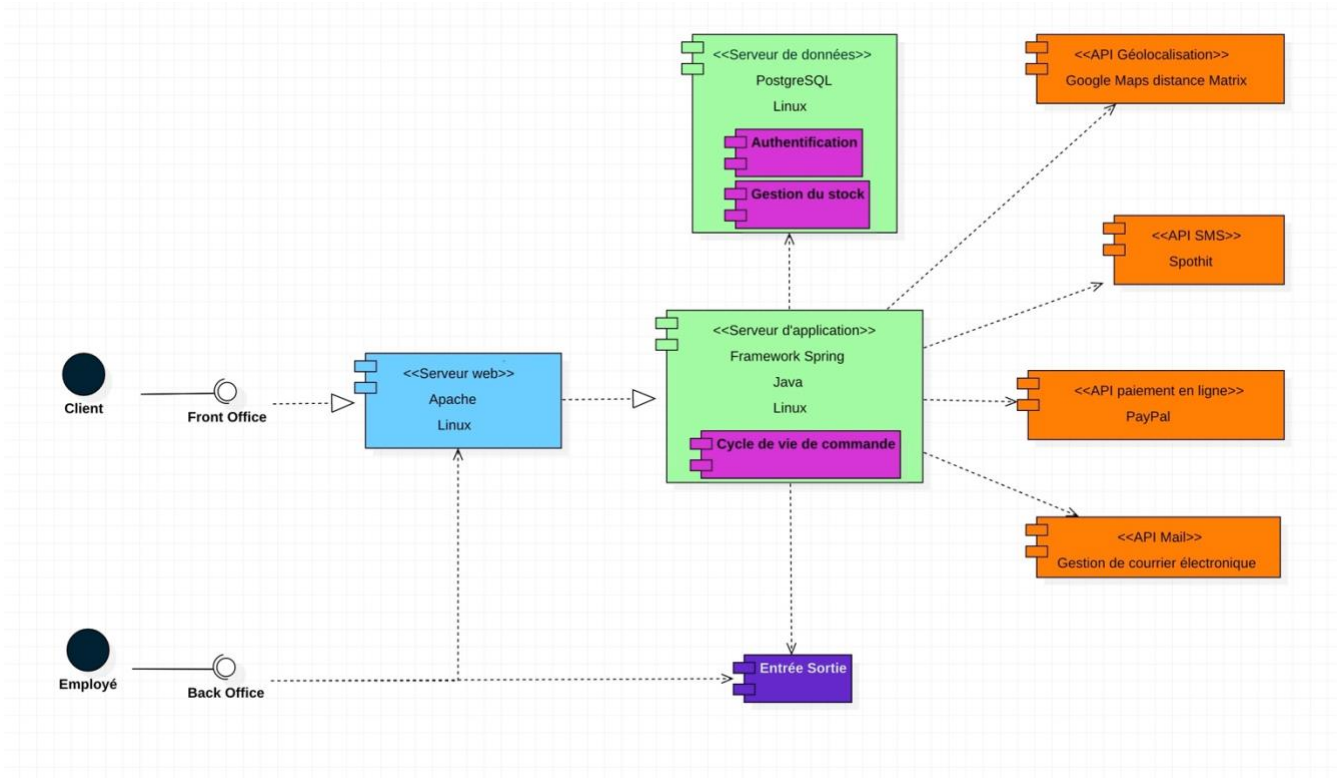
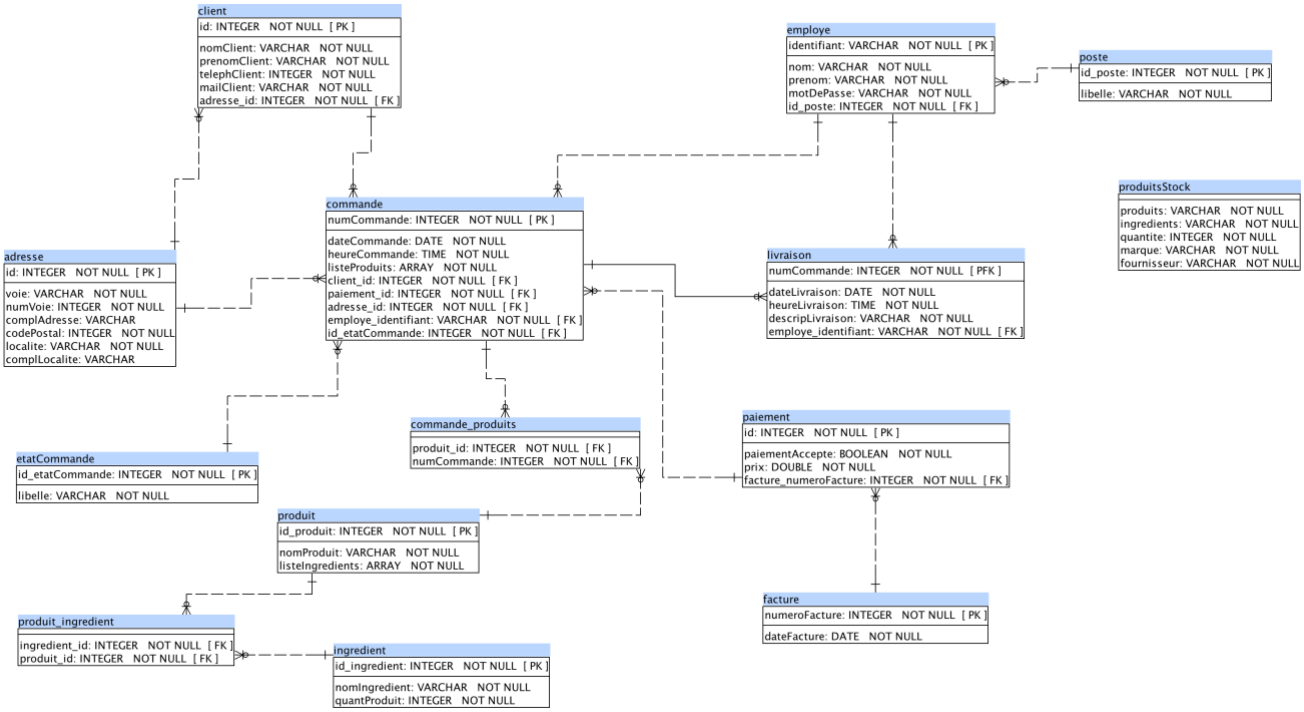


DIAGRAMME 1 : DIAGRAMME DE COMPOSANTS

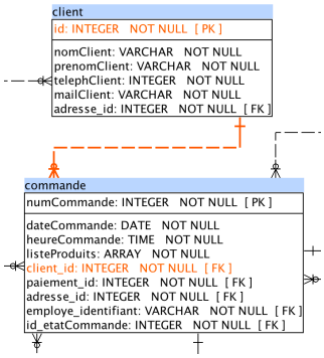
3.2 - Le Modèle Physique de Données

Le modèle physique de données est par excellence la représentation schématique en mesure de générer du code SQL, notamment par l'intermédiaire du logiciel SQL Power Architect, ce code pouvant construire une base de données PostgreSQL. Le modèle physique de données crée pour chacune des classes une clé primaire, celles-ci permettant d'établir des correspondances entre l'ensemble des classes.

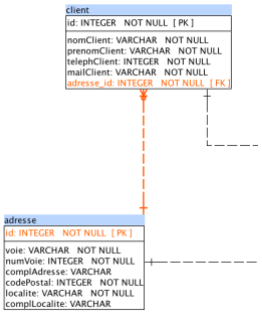


3.2.1 - La relation entre les classes « client » et « commande »

Une commande possède un client (one to one) aussi, la classe « commande » comporte « client_id » en clé étrangère. Cette clé permet d'associer chaque client à une commande unique.

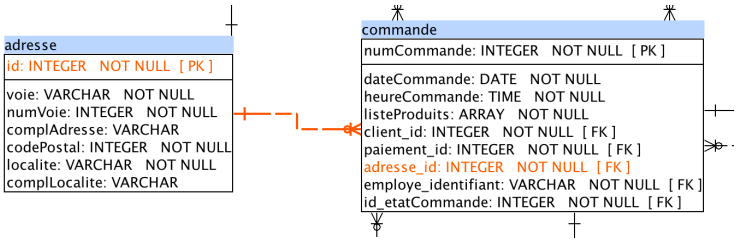


3.2.2 - La relation entre les classes « client » et « adresse »



Chaque client possède une adresse unique. Celle-ci est enregistrée dans la classe « client » via la clé étrangère « adresse_id ».

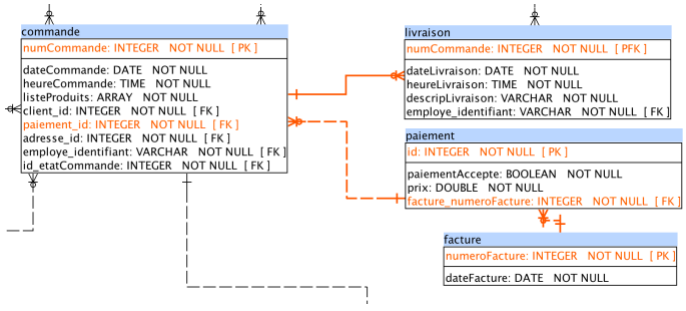
3.2.3 - La relation entre les classes « adresse » et « commande »



Chaque commande est reliée à une adresse unique. Pour ce faire, la classe « commande » possède la clé étrangère « adresse_id ».

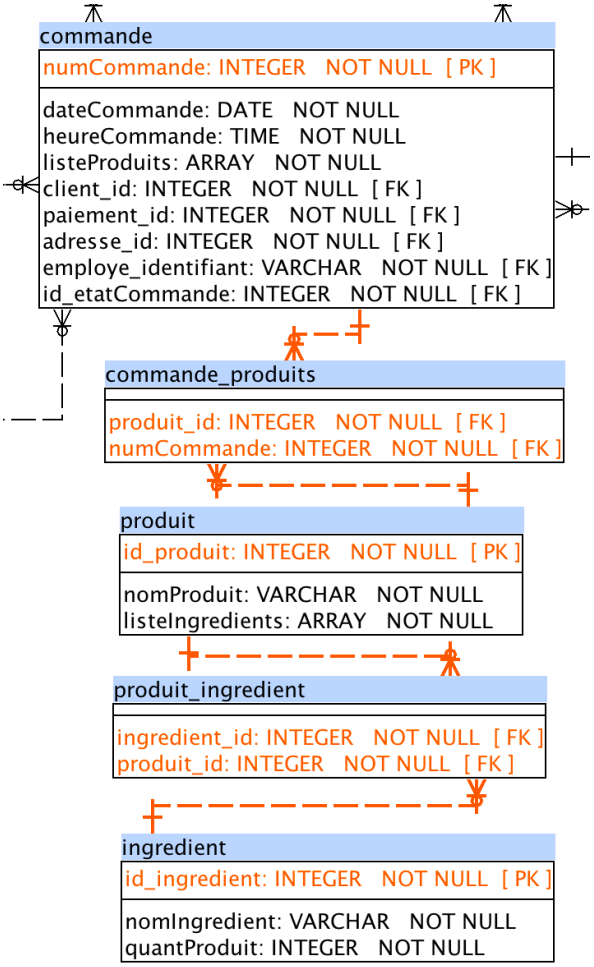
3.2.4 - La relation entre les classes « commande » et « livraison », entre « commande » et « paiement » et entre « paiement » et « facture »

Une commande est en relation avec un paiement en particulier, de même que ce paiement est en référence avec une facture unique.



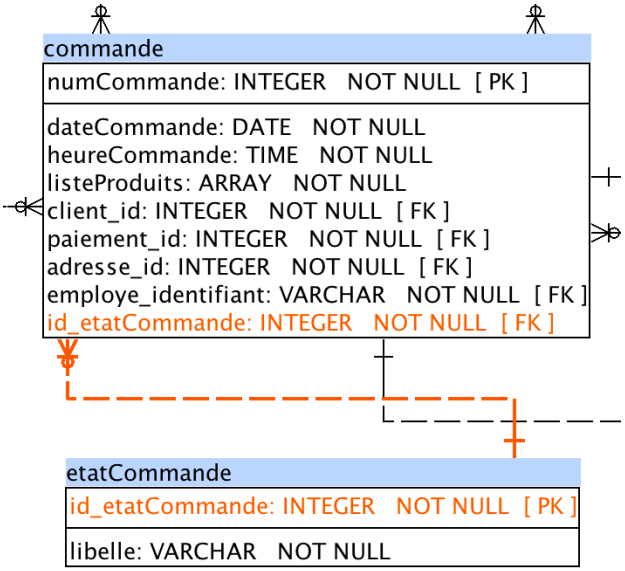
Dans la mesure où une livraison peut correspondre à plusieurs commandes (« one to many »), la classe livraison possède une clé à la fois primaire et étrangère. Celle-ci permet de regrouper les commandes en correspondance avec une livraison unique.

3.2.5 - Les spécificités des relations entre les classes « commande », « produit » et « ingredient »



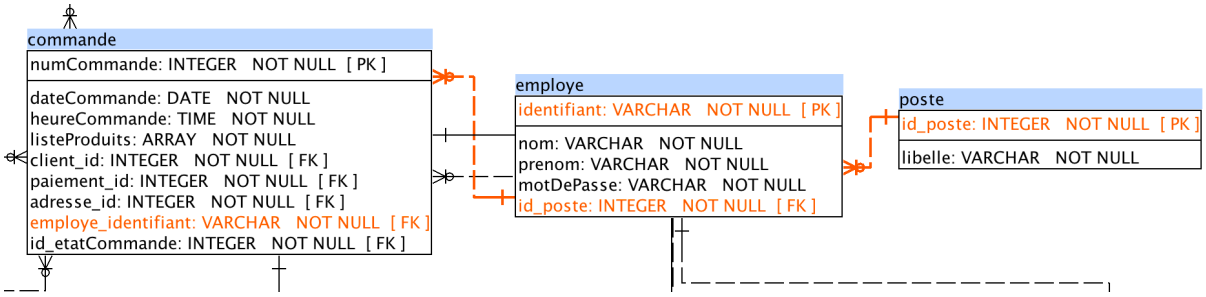
Plusieurs commandes comportent plusieurs produits et plusieurs ingrédients (« many to many »). Afin de mieux structurer les relations entre ces trois tables, il a été nécessaire de créer des tables de liaison : « commande_produits » et « produit_ingredient ». La table de liaison « commande_produits » permet d'enregistrer en clés étrangères les deux clés primaires « numCommande » (clé primaire de la table « commande ») et « produit_id » (clé primaire de la table « produit »). Cette table de liaison définit une première spécificité de la commande client. La seconde table de liaison « produit_ingredient » enregistre en clés étrangères « produit_id » (clé primaire de la table « produit ») et « ingredient_id » (clé primaire de la table « ingredient »). Cette deuxième table de liaison fournit une seconde spécificité de la commande client.

3.2.6 - La relation entre les classes « etatCommande » et « commande »



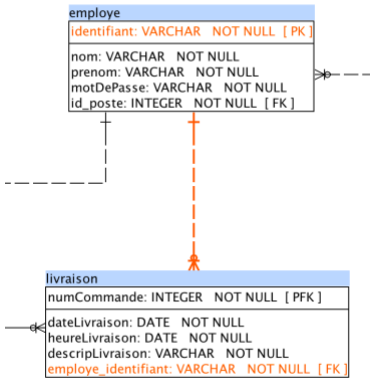
La classe « commande » intègre en clé étrangère « id_etatCommande » afin de spécifier pour chaque commande son état : soit acceptée soit en préparation soit prête ou affectée à un livreur ou en cours de livraison. Cet état permet au client de connaître la situation de sa commande et est spécifié dans l'attribut « libelle ».

3.2.7 - La relation entre les classes « commande », « employe » et « poste »



La classe « employe » intègre la clé étrangère « id_poste ». Cette clé étrangère fait référence à la classe « poste ». Cette classe désigne l'employé en charge de la commande ou de sa livraison. De son côté, la classe « commande » comporte la clé étrangère « employe_identifiant » de manière à récupérer l'identifiant de l'employé désigné en amont.

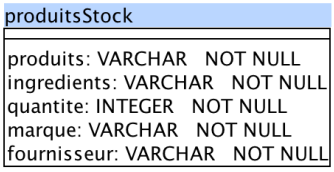
3.2.8 - La relation entre les classes « employe » et « livraison »



La classe « livraison » possède la clé étrangère « employe_identifiant » afin de connaître l'employé en gestion de la commande.

3.2.9 - La classe « produitsStock »

La classe « produitsStock » ne possède aucune clé primaire car celle-ci n'est en relation avec aucune classe. Son unique fonction consiste dans la gestion du stock, de son niveau suffisant et en concordance avec les besoins clients.



4 - Architecture de Deploiement

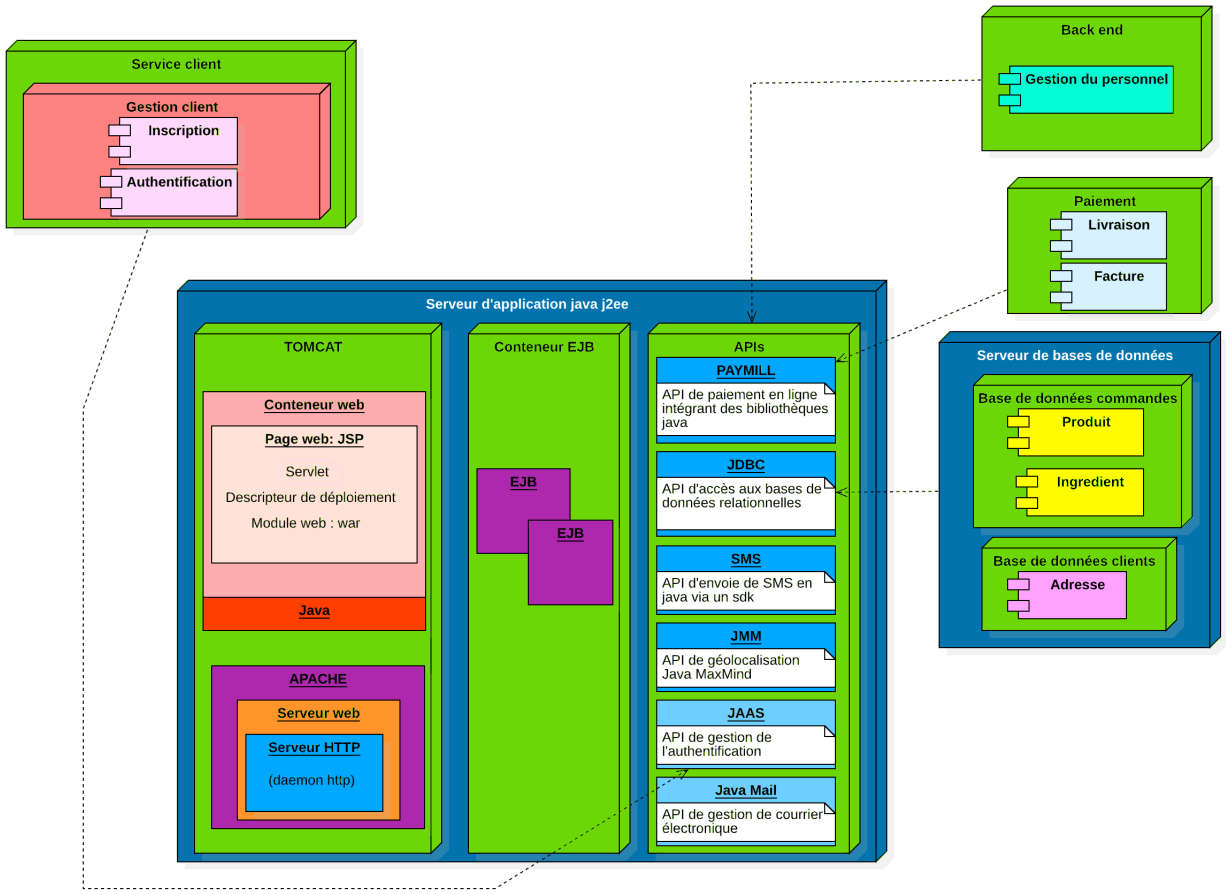
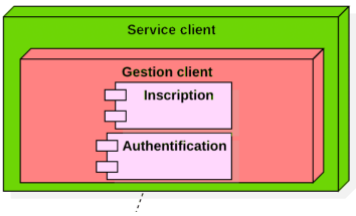


DIAGRAMME 2 : DIAGRAMME DE DÉPLOIEMENT DE L'APPLICATION D'OC PIZZA[1]

4.1 - Le serveur d'application

4.1.1 - L'accès client



Cet accès permet aux clients de se connecter à l'application via une authentification ou une inscription si celle-ci n'a pas déjà été effectuée. Il y a une spécificité d'accès : le client aura un accès un accès restreint à l'application et pourra commander les produits qu'il souhaite.

[1] Extrait de : « Mise en place d'un nouveau système informatique pour l'ensemble des pizzerias du groupe OC Pizza », page 5

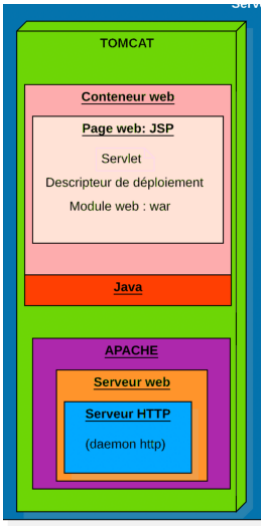
4.1.2 - Le serveur Tomcat

Le serveur regroupe différentes couches, dont :

4.1.2.1 - Le serveur web Http

Ce serveur gère des requêtes HTTP. Il a pour rôle d'intercepter ces requêtes sur le port par défaut 8080, de les traiter et de générer ensuite des réponses Http. Tous les serveurs web embarquent un daemon Http (http) ou équivalent s'occupant de cette fonctionnalité.[2]

4.1.2.2 - Le conteneur Web

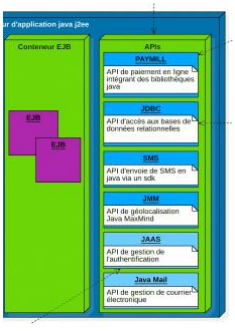


Cette extension va permettre d'avoir la possibilité d'exécuter des programmes écrits avec des langages de programmation (java, PHP, C# ou autres) dans le serveur web : Par exemple le serveur Tomcat n'est autre qu'un serveur Apache couplé avec un moteur web java.[3]

- *Le serveur d'application*

  Il est composé de :

  1. Un conteneur EJB qui encapsule les traitements des Entreprise JavaBeans.
  2. Un ensemble de services d'infrastructures et de communication :
     i) PAYMILL : API de paiement en ligne intégrant des bibliothèques Java.
     ii) JDBC (Java DataBase Connectivity) API d'accès aux bases de données relationnelles.



[2] Source : https://www.supinfo.com/articles/single/1156-difference-serveur-web
[3] Ibid.

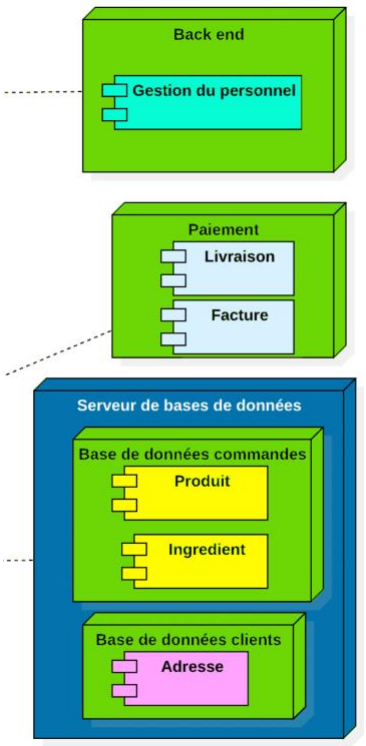
iii) SMS : API d'envoi de SMS en java via un SDK

iv) JMM : API de géolocalisation « Java MaxMind »

v) JAAS (Java Authentication and Authorization Service) : API de gestion de l'authentification.

vi) JavaMail API pour la gestion de courrier électronique.[4]

- Le back-end



Le back-end permet à l'administrateur de gérer les commandes des clients, de les enregistrer et de les traiter. Il permet également aux employés de se connecter à l'application, de prendre connaissance de l'avancée du processus de vente, de l'intercepter si nécessaire, d'effectuer un suivi en temps réel des commandes clients, etc.

[4] Ibid.

5 - ARCHITECTURE LOGICIELLE

5.1 - Principes généraux

Le développement de l'application est un projet basé sur les Framework Spring Boot et Angular. Il est divisé en deux parties :

- Une partie front-end, basée sur le Framework Angular ;
- Une partie back-end, base sur le Framework Spring Boot.

5.1.1 - Les couches

L'architecture de la partie back-end :

- Une couche **business** : responsable de la logique métier du composant ;
- Une couche **consumer** : responsable de l'accès à la base de données
- Une couche **webapp** : applicatif serveur
- Une couche **model** : implémentation du modèle des objets métiers

L'architecture de la partie front-end :

- ***« Des composants web** : la partie visible de l'application Web (IHM) ;*
- ***Des services pour la logique applicative**. Les composants peuvent utiliser les services via le principe de l'injection des dépendances ;*
- ***Les directives** : un composant peut utiliser des directives ;*
- ***Les pipes** : utilisés pour formater l'affichage des données dans les composants. »*[5]

La structuration des répertoires du projet back-end suit la logique suivante :

[5] Source : https://apcpedagogie.com/structure-et-architecture-dun-projet-angular/

Les répertoires sources sont créés de façon à respecter la philosophie Maven (à savoir : « convention plutôt que configuration »)

5.1.1.1 - Projet back-end :

```
racine
├── pom.xml
├── <business>
│   ├── pom.xml
│   └── <src>
│       ├── <main>
│       │   ├── <java>
│       │   │   ├── <dto>
│       │   │   ├── <mapper>
│       │   │   ├── <security>
│       │   │   ├── <service>
│       │   │   │   └── <exception>
│       └── <test>
│           ├── <java>
├── 〈consumer〉
│   ├── pom.xml
│   └── <src>
│       ├── <main>
│       │   ├── <java>
│       │   │   └── <repository>
│       └── <test>
│           ├── <java>
├── 〈webapp〉
│   ├── pom.xml
│   └── <src>
│       ├── <main>
│       │   ├── <java>
│       │   │   ├── <controllers>
│       │   │   ├── <error>
│       │   │   └── Main.java
│       │   └── <resources>
│       │       ├── application.properties
│       │       └── data.sql
│       └── <test>
│           ├── <java>
│           └── <ressources>
├── <model>
│   ├── pom.xml
│   └── <src>
│       ├── <main>
│       │   ├── <java>
│       │   │   └── <beans>
│       │   │       └── <payload>
│       │   │           ├── <request>
│       │   │           └── <response>
│       └── <test>
│           ├── <java>
└── <src>
    └── <lib>
```

5.1.1.2 - Projet front-end :

```
racine
├── angular.json
├── <e2e>
│       ├── protractor.conf.js
│       ├── <src>
│       │       └── appe2e-spec.ts
│       │       └── app.po.ts
│       ├── tsconfig.json
├── karma.conf.js
├── <node_modules>
│       └── __ngcc_entry_points__.json
│       └── <@angular>
│       └── <@angular-devkit>
│       └── <@babel>
│       └── ..................
├── package-lock.json
├── package.json
├── README.md
├── <src>
│       ├── <app>
│       │       └── app-routing.module.ts
│       │       └── app.component.html
│       │       └── app.component.sass
│       │       └── app.component.spec.ts
│       │       └── app.component.ts
│       │       └── app.module.ts
│       │       └── <shared>
│       │               └── index.ts
│       ├── <assets>
│       ├── <environments>
│       │       └── environment.prod.ts
│       │       └── environment.ts
│       ├── favicon.ico
│       ├── index.html
│       ├── main.ts
│       ├── polyfills.ts
│       ├── styles.sass
│       ├── test.ts
├── tsconfig.app.json
├── tsconfig.json
├── tslint.json
```

6 - POINTS PARTICULIERS

6.1 - Gestion des logs

La gestion des logs est assurée à l'aide de Log4j.

6.2 - Fichiers de configuration

6.2.1 - Application back-end

Le fichier application.properties configure le webservice Spring. La base de données y est configurée.

6.2.2 - Datasources

Le fichier data.sql permet de charger, au lancement de l'application back-end, un jeu de données.

6.2.3 - Application front-end

Au démarrage de l'application Angular, un fichier JSON se charge. Ce fichier décrit les paramètres du fichier « environment.ts ».

6.3 - Ressources

Les ressources sont stockées sur le serveur du client Web. Le client web accède à cet espace de stockage afin de charger ces différentes ressources.

6.4 - Environnement de développement

Le Projet 8 a été développé à partir des environnements Spring Boot 2.4.5, Angular et PostGreSQL 12.5 pour la base de données. Angular est associé à Bootstrap dans le but de parfaire l'aspect visuel de l'application.

Si l'administrateur souhaite bénéficier d'une assistance dans la gestion de son stock et donc des commandes, nous pouvons lui proposer une application batch dans le but d'automatiser les commandes des ingrédients et des produits auprès des fournisseurs. Cette application sera « calibrée » sur un « plancher » préalablement défini par l'administrateur ; le batch déclenchera l'exécution de commandes à l'approche de ce plancher, de telle sorte que chacun des stocks d'ingrédients et de produits ne soient jamais en rupture. Ce batch sera développé à l'aide de Spring Batch.

6.5 - Procédure de packaging / livraison

Une fois le développement de l'application achevé, les deux projets pourront être packagés à l'aide de Maven ou à l'aide de npm pour Angular. L'application est versionnée sur GitHub et est disponible à l'adresse suivante : https://github.com/RicoBSJ/Projet_8_OC_DA_Java_J2EE.

La commande « mvn clean package » permet de packager le web service.

Nous pouvons, en fonction des besoins du client, héberger la solution. Dans la positive, l'application sera installée sur nos serveurs sous forme d'un contrat SAAS[6].

Si cette alternative n'intéresse pas le client, nous partagerons le code source de l'application.

[6] Software As A Service