OC PIZZA

Projet 8

Dossier de conception technique
Version 2.0

Auteur
Charlotte Vanhuyse
Développeur d'application junior

TABLE DES MATIÈRES

1 - Versions

Auteur	Date	Description	Version
Charlotte Vanhuyse	04/11/2019	Création du document	1.0
Charlotte Vanhuyse	16/11/2020	Mise à jour document	2.0

2 - INTRODUCTION

2.1 -Objet du document

Le présent document constitue le dossier de conception technique de l'application OC Pizza à l'attention des développeurs, mainteneurs et de l'équipe technique.

Les éléments du présents dossiers découlent de l'analyse des besoins exprimés par le client et de la rédaction du dossier de conception fonctionnelle.

2.2 -Références

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

1. **Projet 8 - Dossier de conception fonctionnelle** : Dossier de conception fonctionnelle de l'application
2. **Projet 8 - Dossier d'exploitation** : Dossier d'exploitation de l'application
3. **Projet8 - procés verbal** : Procès-verbal de l'application

3 - Architecture Technique

3.1 -Diagramme de composants

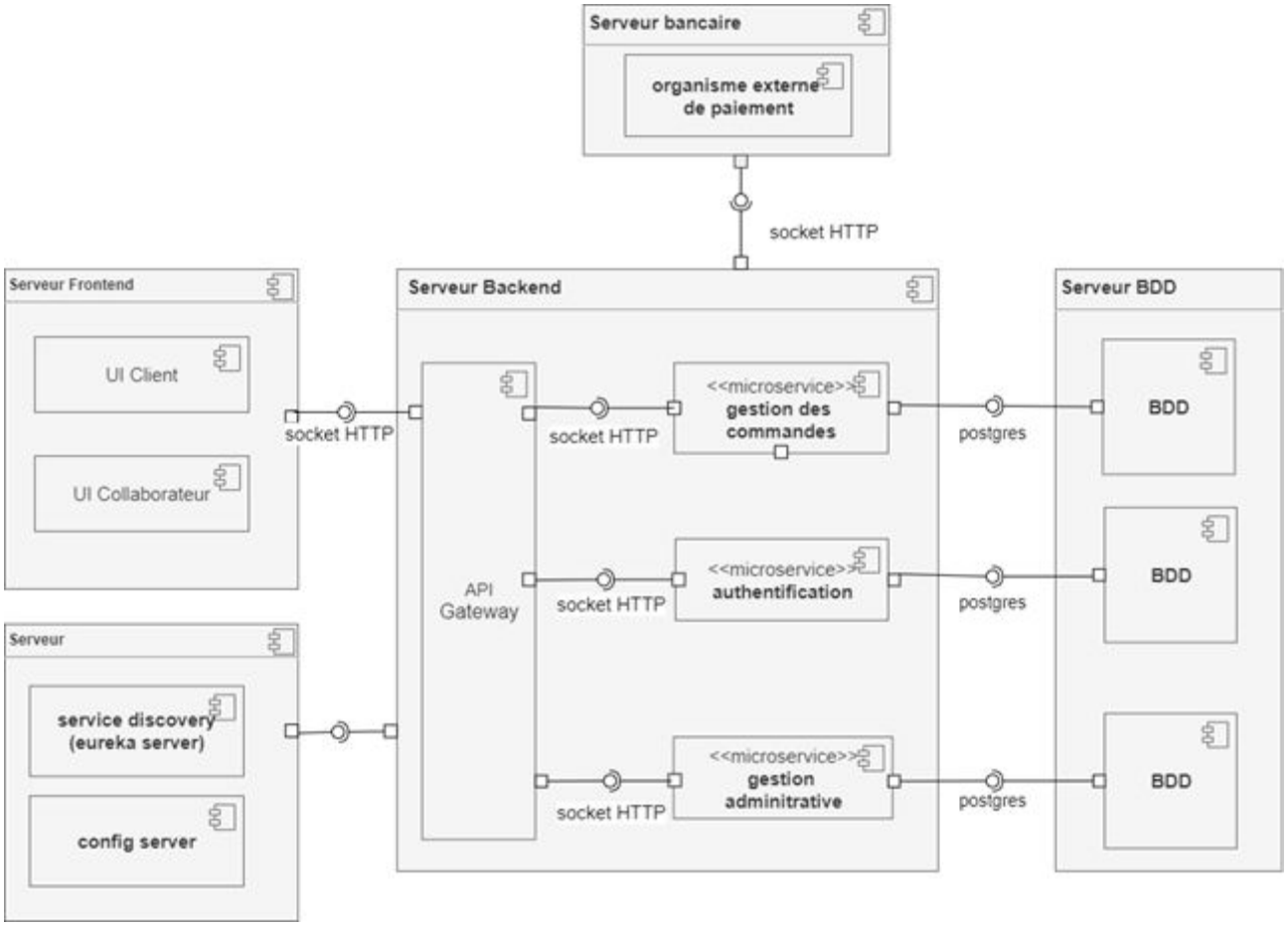


Diagramme de composants

3.2 - Composants généraux

3.2.1 - Frontend

La pile logicielle est la suivante :

- Application web Java (JDK version 1.8)
- SpringBoot (2.3.0.RELEASE)
- Apache Maven (4.0.0)
- IntelliJ Ultimate (2020.2)

3.2.1.1 - Interface Client

Ce composant propose une interface graphique dédiée aux clients.

3.2.1.2 - Interface Collaborateur

Ce composant propose une interface graphique dédiée aux Collaborateurs.

3.2.2 - Backend

3.2.2.1 - Serveur de configuration

Ce composant permet de gérer la configuration de chaque microservice de l'application. On utilisera Config Server qui fait partie de la stack Spring Cloud Netflix.

3.2.2.2 - Serveur Discovery

Ce composant permet d'enregistrer les instances des microservices de l'application. On utilisera Eureka qui fait partie de la stack Spring Cloud Netflix.

3.2.2.3 - API Gateway

Ce composant est l'unique point d'entrée de notre application. On utilisera Zuul Gateway qui fait partie de la stack Spring Cloud Netflix.

3.2.2.4 - Authentification

Ce composant va permettre de s'inscrire, de modifier ses informations, de s'authentifier et de vérifier les autorisations des utilisateurs.

3.2.2.5 - Gestion des commandes

Ce composant permet de gérer les commandes de l'achat à la livraison.

3.2.2.6 - Gestion administrative

Ce composant permet de gérer les stocks, les produits proposés, les recettes. Il permet d'analyser l'activité des pizzerias et aussi de gérer les comptes utilisateur de l'application.

3.2.3 - Base de données

3.2.3.1 - BDD Authentification

Ce composant correspond à la base de données de l'application qui communique avec le microservice "Authentification".

3.2.3.2 - BDD Gestion des commandes

Ce composant correspond à la base de données de l'application qui communique avec le microservice "Gestion des commandes".

3.2.3.3 - BDD Gestion administrative

Ce composant correspond à la base de données de l'application qui communique avec le microservice "Gestion administrative".

3.2.4 - Composant externe

3.2.4.1 - Service bancaire

L'organisme externe de paiement fournit l'interface qui permettra d'effectuer des paiements via l'application.

4 - Architecture de Déploiement

4.1 -Diagramme de déploiement

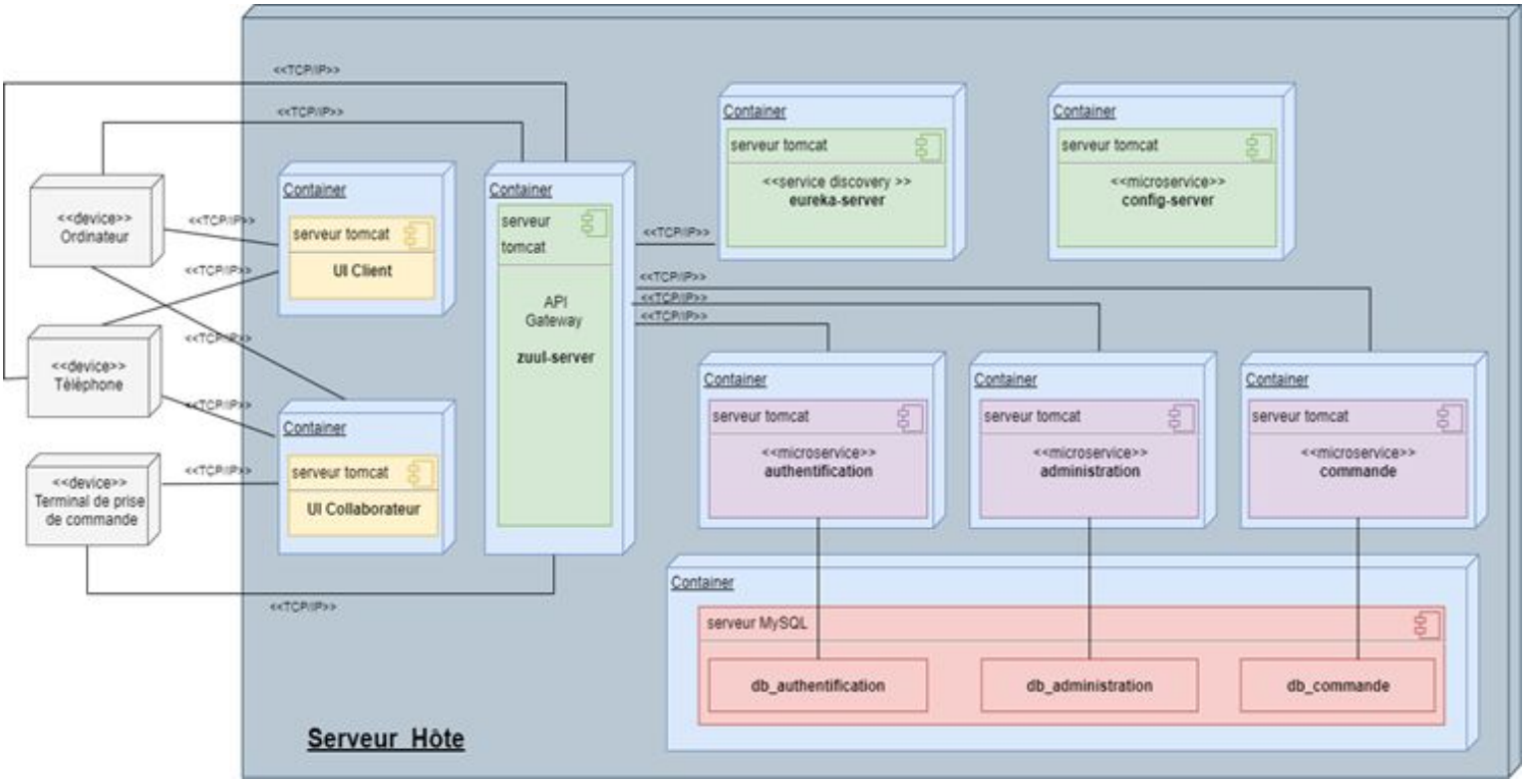


diagramme de déploiement

4.2 -Serveur dédié

L'application sera déployée sur un serveur dédié de AWS qui distribue un noyau Linux 4..14, on utilisera Amazon Linux 2 AMI.

L'application sera déployé via docker-compose :

- Chaque microservice est dockerisé dans un conteneur qui lui est propre, le service est déployé sur un serveur Tomcat inclus avec le framework SpringBoot.
- Les BDD sont dockerisés dans un seul conteneur, on utilisera un serveur MySQL.

5 - ARCHITECTURE LOGICIELLE

5.1 -Principes généraux

Les sources et versions du projet sont gérées par **Git**, Le code source de l'application se trouve sur **Github**, les dépendances et le packaging seront gérés par **Apache Maven**.

Le développement de l'application se fera en suivant une architecture microservice, on utilisera le frawework SpringBoot.

5.1.1 - Les couches

L'architecture applicative est la suivante :

- une couche **business** : responsable de la logique métier du composant, elle est implémentée dans le package "service".
- une couche **model** : implémentation du modèle des objets métiers, elle est implémentée dans le package "entity".
- une couche **consumer** : responsable des interactions avec la base de données, elle est implémentée dans le package "repository" qui comporte les DAO.
- une couche **présentation** : constitue l'interface entre le client et l'application, elle est implémentée dans le package "controler".
- un package "**configuration**" sera également présent.

5.2 -Api Web

5.2.1 - Les modules

L'application suit une architecture multi-module Maven où chaque module est dédié à un microservice différent :

- Config-server
- Eureka-server
- Api-gateway
- Authentification
- Gestion des commandes
- Gestion administrative.

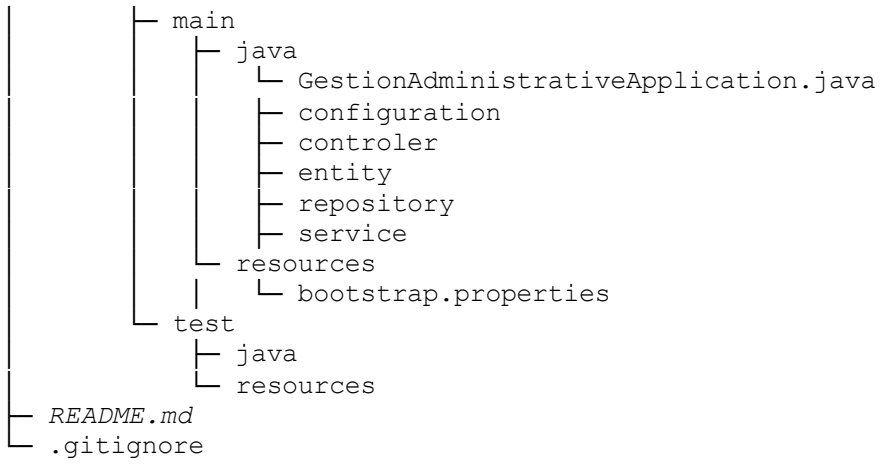
5.2.2 - Structure des sources

La structuration des répertoires du projet suit la logique suivante :

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

```
ocpizza
├─ pom.xml
├─ docker-compose.yml
├─ <config-server>
│   ├─ pom.xml
│   ├─ Dockerfile
│   ├─ .gitignore
│   └─ src
│       ├─ main
│       │   ├─ java
│       │   │   ├─ configuration
│       │   │   ├─ controler
│       │   │   ├─ entity
│       │   │   ├─ repository
│       │   │   ├─ service
│       │   │   └─ ConfigServerApplication.java
│       │   ├─ resources
│       │   │   └─ config.properties
│       └─ test
│           ├─ java
│           └─ resources
├─ <eureka-server>
│   ├─ pom.xml
│   ├─ Dockerfile
│   ├─ .gitignore
│       ├─ main
│       │   ├─ java
│       │   │   ├─ configuration
│       │   │   ├─ controler
│       │   │   ├─ entity
│       │   │   ├─ repository
│       │   │   ├─ service
│       │   │   └─ EurekaServerApplication.java
│       │   ├─ resources
│       │   │   └─ bootstrap.properties
│       └─ test
│           ├─ java
│           └─ resources
├─ <api-gateway>
│   ├─ pom.xml
│   ├─ Dockerfile
│   ├─ .gitignore
│   └─ src
│       ├─ main
│       │   ├─ java
```

```
│           │   │   ├── configuration
│           │   │   ├── controler
│           │   │   ├── entity
│           │   │   ├── repository
│           │   │   ├── service
│           │   │   └── ApiGatewayApplication.java
│           │   ├── resources
│           │   │   └── bootstrap.properties
│           └── test
│               ├── java
│               └── resources
├── <authentification>
│   ├── pom.xml
│   ├── Dockerfile
│   ├── .gitignore
│   └── src
│       ├── main
│       │   ├── java
│       │   │   ├── configuration
│       │   │   ├── controler
│       │   │   ├── entity
│       │   │   ├── repository
│       │   │   ├── service
│       │   │   └── AuthentificationApplication.java
│       │   ├── resources
│       │   │   └── bootstrap.properties
│       └── test
│           ├── java
│           └── resources
├── <gestion-commandes>
│   ├── pom.xml
│   ├── Dockerfile
│   ├── .gitignore
│   └── src
│       ├── main
│       │   ├── java
│       │   │   ├── configuration
│       │   │   ├── controler
│       │   │   ├── entity
│       │   │   ├── repository
│       │   │   ├── service
│       │   │   └── GestionCommandeApplication.java
│       │   ├── resources
│       │   │   └── bootstrap.properties
│       └── test
│           ├── java
│           └── resources
├── <gestion-administrative>
│   ├── pom.xml
│   ├── Dockerfile
│   ├── .gitignore
│   └── src
```


main
java
GestionAdministrativeApplication.java
configuration
controler
entity
repository
service
resources
bootstrap.properties
test
java
resources
README.md
.gitignore

5.3 -Application Web

5.3.1 - Les modules

Deux applications web sont clientes de l'API Web :

- UI-client : elle sera utilisée par les clients des pizzerias.
- UI-collaborateur : elle sera utilisée par les collaborateurs des pizzerias.

5.3.2 - Structures des sources

```
<ui-client>
   ├─ pom.xml
   ├─ Dockerfile
   ├─ .gitignore
   └─ src
      └─main
          ├─ java
          │   ├─ configuration
          │   ├─ controler
          │   ├─ entity
          │   ├─ proxy
          │   └─ UIClient.java
          ├─ resources
          │   ├─ templates
          │   └─ bootstrap.properties
          └─ test
              ├─ java
              └─ resources

<ui-collaborateur>
   ├─ pom.xml
   ├─ Dockerfile
   ├─ .gitignore
   └─ src
      └─main
          ├─ java
          │   ├─ configuration
          │   ├─ controler
          │   ├─ entity
          │   ├─ proxy
          │   └─ UICollaborateur.java
          ├─ resources
          │   ├─ templates
          │   └─ bootstrap.properties
          └─ test
              ├─ java
              └─ resources
```

6 - Points particuliers

6.1 -Gestion des logs

Logback est nativement incorporé dans **SpringBoot** via la dépendance **spring-boot-starter**, elle-même comprise dans toutes les dépendances de type starter. LogBack utilise **SLF4J** comme interface native.

6.2 -Fichiers de configuration

On utilise un serveur de configuration pour centraliser les propriétés de chaque microservice : **config-server**.

Dans le fichier "**config.properties**" du module config-server sera indiqué l'url du repository Github utilisé par défaut.

```
spring.cloud.config.server.git.uri  =
${SPRING_CONFIG_SERVER_URI:https://github.com/vancharlotte/config-server-ocpizza.git}
```

Chaque microservice a un fichier "**bootstrap.properties**" dans lequel est présent l'uri nécessaire pour récupérer sa configuration.

```
Spring.application.name = nomdumicroservice

spring.cloud.config.uri = ${SPRING_CONFIG_URI:http://config-server:9101}
```

Les variables d'environnement utilisées dans les fichiers de configuration sont directement modifiables dans le fichier "**docker-compose.yml**" présent à la racine du projet.

6.3 -Environnement de développement

L'IDE utilisé par nos équipes est intelliJ IDEA Ultimate 2020.2.

6.4 -Procédure de packaging / livraison

Le code source de l'application est disponible sur Github :

```
https://github.com/vancharlotte/projet8-ocpizza.git
```

Pour pouvoir packager et déployer l'application sur le serveur dédié, vous pouvez vous référer au dossier d'exploitation.

7 - GLOSSAIRE

AMI	Amazon Machine Image
AWS	Amazon Web Service
BDD	Base de données
DAO	Data Access Object
IDE	Environnement de Développement Intégré
SLF4J	Simple Logging Facade for Java
UI	Interface Utilisateur