OC Pizza

PizzaFlow

Dossier de conception technique

Version 1.0

Auteur
Paul-Emmanuel DOS SANTOS FACAO
Analyste programmeur

TABLE DES MATIÈRES

1 - VERSIONS

Auteur	Date	Description	Version
PEDSF	06/05/2020	Création du document	1.0

2 - INTRODUCTION

2.1 - Objet du document

Le présent document constitue le dossier de conception technique de l'application **PizzaFlow** à l'attention des développeurs, mainteneurs et de l'équipe technique du client.

Les éléments du présent dossier découlent :

- Des besoins exprimés par le client **OC Pizza** lors du premier contact,
- De l'analyse des besoins de **OC Pizza**,
- De la rédaction du dossier de conception fonctionnelle.

2.2 - Références

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

1. **DCT - PDOCPizza_01_fonctionnelle** : Dossier de conception fonctionnelle de l'application.
2. **DCT - PDOCPizza_03_exploitation** : Dossier d'exploitation de l'application.

3 - Architecture Technique

3.1 - Composants généraux

Pour faciliter la maintenance, la robustesse, la sécurité, la scalabilité et le développement, on utilise une architecture à base de microservices. Pour réduire les coûts d'investissement de départ et facilité la monté en puissance du système, on utilise **AWS** pour déployer **PizzaFlow**. En outre AWS fourni de nombreux services annexes pour la mise en œuvre du système.

3.1.1 - Le serveur de configuration

3.1.1.1 - config-server

Il est indispensable de centraliser les configurations pour être sûr que plusieurs instances d'un même microservice utilise la même configuration. Cela permet aussi d'avoir un seul changement de paramètre pour plusieurs instances d'un microservice. Pour l'implémentation on utilisera **spring-cloud-config-server** pour le serveur et **spring-cloud-config-client** pour les microservices. Chaque microservice demandera sa configuration à **config-server**. La pile logicielle est la suivante :

- Application: **J2EE** (**JDK** 1.8) / **Apache Maven** (3.6.3)/ **Spring** (2.2.6.RELEASE)

3.1.1.2 - Repository

Les différentes configurations des microservices seront dans un repository privé pour en faciliter l'accès. Dans tous les cas, il faudra crypter les données sensibles des fichiers properties avec **Jasypt**. Seules les personnes autorisées auront accès à la clé de cryptage pour encoder les données.

3.1.2 - La gateway

Pour publier, gérer, surveiller et sécuriser facilement les **API** du système d'information on utilise le service **AWS API Gateway**. En liaison avec **AWS IAM** (**Identity and Access Management**) pour effectuer l'identification des utilisateurs.

3.1.3 - Le load-balancer

Avec l'accroissement du nombre de clients sur internet, certaines **APIs** seront dupliquées pour répartir la charge. Un **AWS Elastic Load Balancing** se charge de répartir la charge sur les différentes

instances d'une **API**.

3.1.4 - Un bucket

On utilisera une instance **AWS S3** (**Simple storage Service**) pour stocker les données statiques comme les photos des produits et les fichiers **HTML**, **CSS** et **JS**.

3.1.5 - Package Utilisateur

3.1.5.1 - user-api

C'est un contrôleur RESTful qui communique directement avec la base de données Utilisateur. Il implémente toutes les opérations de gestion des utilisateurs qu'ils soient clients ou employé d'**OC Pizza**. C'est la partie backend qui fait l'intermédiaire entre le frontend et la base de données. Les différentes opérations sont accessibles suivant l'authentification et les droits de l'utilisateur. Seules les personnes habilitées pourront créer des comptes pour les Employés. Le Client pourra créer et modifier son compte mais pas le supprimer. L'Employé aura aussi le droit de créer un compte pour un Client lors d'une commande en magasin pour une personne non inscrite. C'est cette **API** qui sera en charge du paiement bancaire pour limiter le transit de données sensibles. La pile logicielle est la suivante :

- Application: **J2EE** (**JDK** 1.8) / **Apache Maven** (3.6.3)/ **Spring** (2.2.6.RELEASE)

3.1.5.2 - Base de données utilisateurs

Pour des soucis de sécurité, de confidentialité et suivant les obligations de la CNIL il est préférable de séparer les données des utilisateurs des autres données. Cette base de données contiendra en plus des identifiants et des coordonnées des Clients, les indications bancaires pour les paiements par internet. Une implantation de la base de données utilisateur au sein des locaux de OC Pizza peut plus facilement avoir l'aval de la CNIL en sécurisant les accès des machines et des locaux où elles sont. La pile logicielle est la suivante :

- **SGBD-R** : **PostgreSQL** (12)

3.1.6 - Package Stock

3.1.6.1 - stock-api

Cette API va concentrer les opérations sur les stocks des magasins. Ce contrôleur RESTful recevra les opérations de décompte de stock lors de la validation des paniers ainsi que celles addition lors de réception d'un bon de livraison d'un fournisseur pour un magasin. Il doit gérer les produits composés comme les pizzas ou les caisses de boissons ainsi que les différentes unités. Suivant le stock d'un magasin, il doit déterminer si une pizza peut être fabriquer et mise en vente. Il faut prévoir des indicateurs sur les produits de base pour savoir s'il faut en commander. Pour les produits achetés, on doit avoir l'information du prix d'achat suivant l'unité indiquée. Pour les produits vendus ont doit renseigner le prix de vente hors taxe et les taux de taxe pour la vente à emporté et la vente livrée. La pile logicielle est la suivante :

- Application: **J2EE** (**JDK** 1.8) / **Apache Maven** (3.6.3)/ **Spring** (2.2.6.RELEASE)

3.1.6.2 - Base de données stock

Elle est accessible par le contrôleur stock-api. Les données n'ont pas besoin d'être aussi sécurisé que les données utilisateurs. La pile logicielle est la suivante :

- **SGBD-R** : **PostgreSQL** (12)

3.2 - Application web

3.2.1 - Package Vente

3.2.1.1 - web-api

C'est l'application utilisée par les Clients et les Employés pour effectuer les commandes. C'est la partie frontend de **PizzaFlow**. Elle interagit avec user-api pour créer ou sélectionner les clients et stock-api pour afficher les produits et modifier le stock. Une interface avec le système bancaire permet d'effectuer les autorisations et valider les paiements par carte bancaire ou chèque. La pile logicielle est la suivante :

- Application : **J2EE** (**JDK** 1.8) / **Apache Maven** (3.6.3)/ **Spring** (2.2.6.RELEASE) / **HTML** (5) / **CSS** (3) / **Thymeleaf** (3.0.11.RELEASE) / **Bootstrap** (4.4)

3.2.1.2 - Base de données ventes

Elle est accessible par l'API web pour enregistrer les commandes et les paniers en cours. Les données n'ont pas besoin d'être aussi sécurisé que les données utilisateurs. La pile logicielle est la suivante :

- **SGBD-R** : **PostgreSQL** (12)

3.2.2 - Package Production

3.2.2.1 - prod-api

C'est avec cette application Web que les membres du personnel interagissent avec **PizzaFlow**. Le Pizzaiolo peut sélectionner une commande en attente, consulter les recettes et indiquer la fin de la fabrication. Le Livreur peut voir les commandes en attente d'être livrés, sélectionner une commande pour partir en livraison, consulter les indications de l'adresse, enregistrer le règlement si nécessaire pour terminer la livraison.

Cette API interagi avec vente-api pour avoir la liste des commandes en cours et effectuer les modifications des statuts des commandes et enregistrer les paiements lors de la livraison.

C'est une application Web simplifiée qui envoie des requêtes et n'a pas d'interaction directe avec les bases de données. La pile logicielle est la suivante :

- Application : **J2EE** (**JDK** 1.8) / **Apache Maven** (3.6.3)/ **Spring** (2.2.6.RELEASE) / / **HTML** (5) / **CSS** (3) / **Thymeleaf** (3.0.11.RELEASE) / **Bootstrap** (4.4)

3.2.3 - Package Gestion

3.2.3.1 - gestion-api

Elle récupère les données de vente du package vente dans sa propre base de données pour pouvoir effectuer des analyses sur les ventes. Cette application étant accessible uniquement par le Manager et la Direction, c'est via celle-ci que l'on effectuera la gestion des clients autre que la création. Avec l'accroissement du nombre de magasins il faut prévoir la mise en place d'une vraie BI avec une base de données propre de type **NoSQL**. La pile logicielle est la suivante :

- Application : **J2EE** (**JDK** 1.8) / **Apache Maven** (3.6.3)/ **Spring Boot** (2.2.6.RELEASE) / **Thymeleaf** (3.0.11.RELEASE) / **Bootstrap** (4.4)

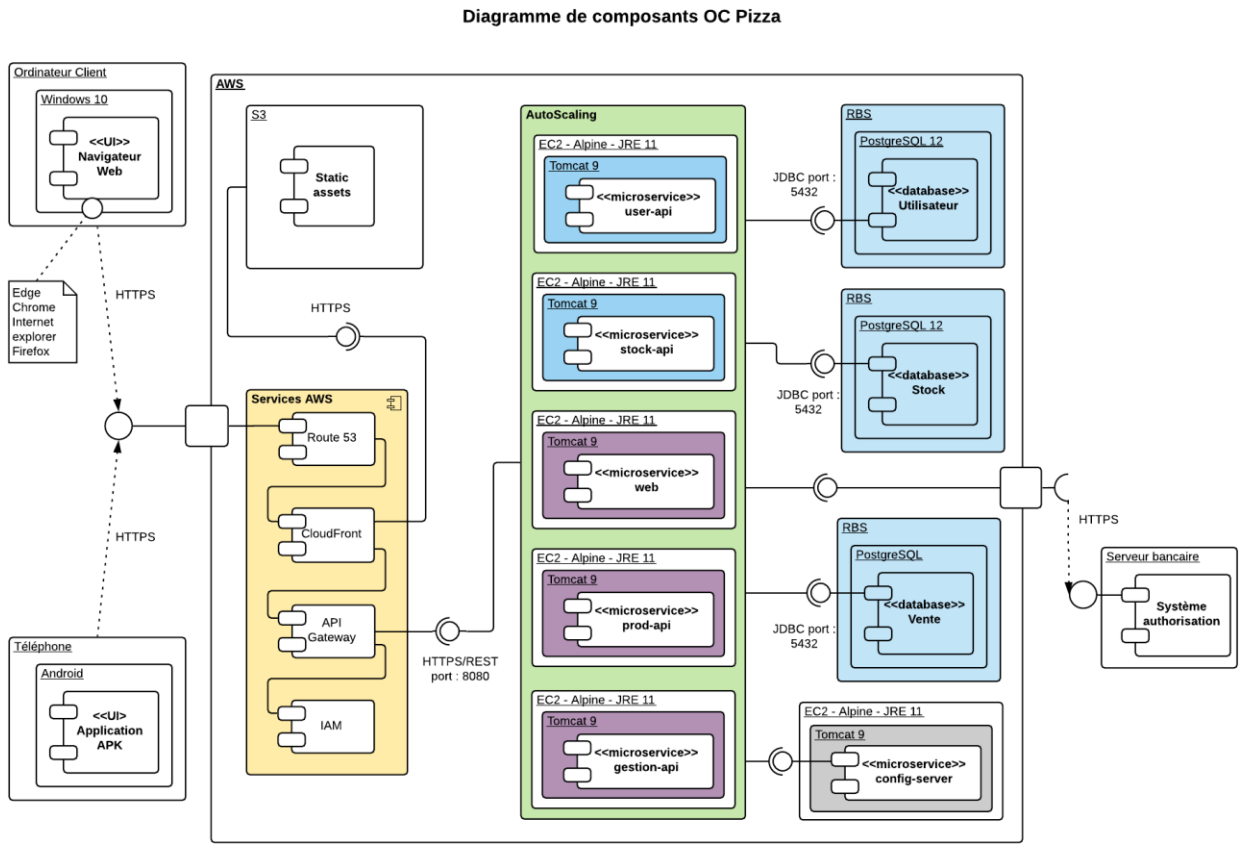
3.2.3.2 - Base de données gestion

Cette base ne contenant pas d'information sensible sur les clients, elle n'a donc pas besoin d'être

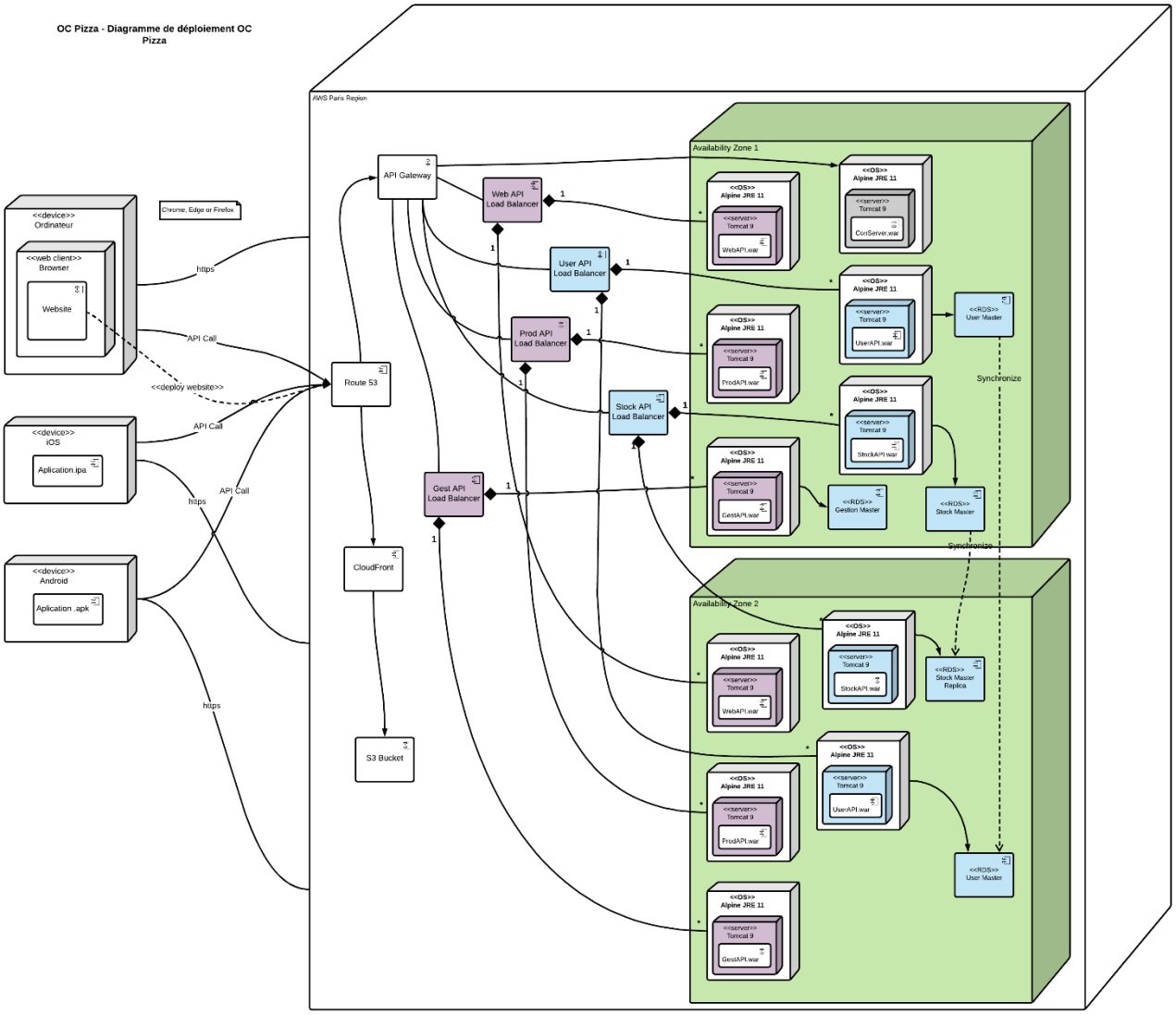
sécuriser comme celle des données utilisateurs. On peut commencer avec une base **SQL** commune avec la base de données Vente et enregistrer exhaustivement les données de vente pour pouvoir faire les statistiques est analyses sans perturber la base de données des commandes. Avec l'accroissement du nombre de magasin et des données de vente, il faut prévoir de basculer vers une base de données **NoSQL** comme Cassandra. La pile logicielle est la suivante :

- **SGBD-R** : **PostgreSQL** (12)

3.2.4 - Diagramme de composants



4 - Architecture de Déploiement



On sépare la partie frontend en trois parties :

- Web pour la vente sur internet et en magasin.
- Production-api pour l'utilisation interne par les pizzaiolos et les livreurs.
- Gestion-api pour tout ce qui est gestion, commande, suivi des ventes en utilisation exclusive par le manager et la direction. C'est la seule API qui accède directement à sa base de données.

On retrouve notre serveur de configuration config-server. Et les APIs user-api et stock-api font l'intermédiaire entre leur base de données et les autres microservices.

L'application est scalable pour répondre à la demande et des load-balancers répartissent la charge sur les différentes instances d'u même microservice.

AWS permet de spécifier des planificateurs pour le fonctionnement des microservices afin de moduler les instances en fonction de la demande.

On peut aussi répartir les microservices sur plusieurs zones pour répartir le trafic internet et s'étendre à l'international.

4.1 - Serveur de Base de données

Pour déployer les bases de données on utilise le service **Amazon RDS** for **PostgreSQL** qui intègre une base de données **PostgreSQL** version **11**.
Comme indiqué plus haut dans ce document, par mesure de sécurité on utilisera une base pour la gestion des utilisateurs.
La base des stocks pourra contenir en outre les informations sur les paniers et les commandes en cours. Par économie on pourra mettre sur le même serveur la base du package gestion.
Pour mieux sécuriser les données et augmenter la réponse des bases de données, on utilisera une base maître et des réplicas qui seront uniquement accessible pour des requêtes de lecture.

4.2 - Serveur de configuration

Il est unique et est déployé sur une instance **EC2** (**Elastic Cloud Compute**). On utilise comme OS une Linux **AMI** (**Amazon Machine Image**) version 2018.03.0 HVM (64bits) qui comporte les outils Java pour exécuter le JAR du serveur :

- **java** version (1.7.0_251)
- **OpenJDK Runtime Environment** (amzn-2.6.21.0.82.amzn1-x86_64 u251-b02)
- **OpenJDK 64-Bit Server VM** (build 24.251-b02, mixed mode)

4.3 - Serveur de fichier

On utilisera une instance **S3** (**Simple Storage Service**) pour stocker les fichiers statiques des pages web et les images utilisés par les applications **web**, **production-api** et **gestion-api**. On peut utiliser Amazon **S3** pour stocker et récupérer n'importe quelle quantité de données, n'importe quand et depuis n'importe quel emplacement sur le Web.

4.4 - Microservices

Chaque microservice sera déployé comme le serveur de configuration sur une instance **EC2**. On utilisera Linux **AMI** version 2018.03.0 HVM (64bits) qui comporte les outils Java :

- **java** version (1.7.0_251)
- **OpenJDK Runtime Environment** (amzn-2.6.21.0.82.amzn1-x86_64 u251-b02)
- **OpenJDK 64-Bit Server VM** (build 24.251-b02, mixed mode)

5 - ARCHITECTURE LOGICIELLE

5.1 - Principes généraux

Le versionnage du projet est effectué avec **Git Flow** et le code source est conservé sur repository privé **GitLab**. Le code est implémenté en **Java2EE**, les dépendances et le packaging sont effectués par **Apache Maven**. On utilise le Framework open source **Spring** pour construire et définir l'infrastructure de l'application. On divise l'application en deux types de microservice sans compter le serveur de fichiers de configuration.

5.1.1 - Les couches

Deux couches sont communes aux microservices qui sont en liaison directe avec les bases de données comme **user-api**, **stock-api** et **gestion-api**.

- La couche **model** qui contient les modèles des objets métiers (**Entity**).
- La couche **business** qui fait la liaison entre le **controller** et la base de données en manipulant les **DTOs** et les **Entitys**.

La couche de présentation est commune aux **API** Web comme **web-api**, **production-api** et **gestion-api**.

- La couche **vue** qui constitue l'interface entre l'utilisateur et l'application et permet de saisir et afficher les données des **DTO**.

Les couches communes à tous les microservices sont :

- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** aux autres **API**, consomme et produit des **DTO** sous forme de **JSON** (**JavaScript Object Notation**).
- La couche **controller** qui expose des **endpoints** et permet d'exécuter les opérations déclenchées par la couche **vue**.

5.1.2 - Les modules

On utilise **Apache Maven** pour le packaging de l'application. Chaque microservice constitue un module de l'application. En ajoutant le serveur de configuration on obtient les modules suivants :

- **config-server** : pour le serveur de configuration.
- **user-api** : pour le contrôleur de la base de données Utilisateur.

- **stock-api** : pour le contrôleur de la base de données Stock.
- **web-api** : pour l'application de vente.
- **production-api** : pour l'application de production.
- **gestion-api** : pour l'application de gestion.

5.1.3 - Structure des sources

Pour alléger les diagrammes, les sous répertoires test ne seront pas représentés. La structure des répertoires **test/java** est la même que celle des répertoires **main/java** de chaque module. Un fichier **README.md** décrira le contenu de chaque module. Le fichier **application.properties** dans les dossiers **test/resources** contiennent toutes les propriétés nécessaires à la phase de test. Chaque module aura son fichier de configuration pom.xml pour **Apache Maven** et son fichier d'intégration **gitlab-ci.yml** pour **GitLab**. La structuration des répertoires du projet suit la logique suivante de façon à respecter la philosophie **Maven**. On aura un répertoire par module et les fichiers statics pour les applications seront centralisés au même endroit. Le repository sera dans un sous répertoire de **config-server**. Le fichier pom.xml principal contiendra toutes les déclarations des versions pour les dépendances.

```
PizzaFlow
├─ <config-server>
│   └─ properties-repository
├─ <user-api>
├─ <stock-api>
├─ <web-api>
├─ <production-api>
├─ <gestion-api>
├─ static
│   ├─ css
│   ├─ images
│   └─ js
│
├─ pom.xml
└─ README.md
```

5.2 - config-server

5.2.1 - Structure des sources

```
├─ <config-server>
│   ├─ properties-repository
│   ├─ src
```

```
│   │   ├── main
│   │   │   ├── java
│   │   │   │   ├── com.pizzaflow.configserver
│   │   │   │   └── ConfigServerApplication.java
│   │   │   └── resources
│   │   │       └── application.properties
│   │   └── test
│   │       ├── java
│   │       └── resources
│   │           └── application.properties
│   │
│   ├── .gitignore
│   ├── .gitlab-ci.yml
│   ├── pom.xml
│   └── README.md
```

5.3 - user-api

5.3.1 - Les couches

Les microservices backend sont des contrôleurs **RESTful** avec lesquels les autres API communiquent. Ils sont découpés suivant les couches suivantes :

- La couche **model** qui contient les modèles des objets métiers (Entity).
- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** aux autres **API**, consomme et produit des **DTO** sous forme de **JSON** (**JavaScript Object Notation**).
- La couche **business** qui fait la liaison entre le **controller** et la base de données en manipulant les **DTOs** et les **Entitys**.

5.3.2 - Structure des sources

```
├── <user-api>
│   ├── src
│   │   ├── main
│   │   │   ├── java
│   │   │   │   └── com.pizzaflow.userapi
│   │   │   │       ├── configuration
│   │   │   │       ├── controller
│   │   │   │       ├── dto
```

```
│   │   │   │       ├─ exception
│   │   │   │       ├─ model
│   │   │   │       ├─ repository
│   │   │   │       ├─ service
│   │   │   │       └─ UserApiApplication.java
│   │   │   └─ resources
│   │   │      └─ bootstrap.properties
│   │   └─ test
│   │      ├─ java
│   │      └─ ressources
│   │         └─ application.properties
│   │
│   ├─ .gitignore
│   ├─ .gitlab-ci.yml
│   ├─ pom.xml
│   └─ README.md
```

5.4 - stock-api

5.4.1 - Les couches

Les microservices backend sont des contrôleurs **RESTful** avec lesquels les autres API communiquent. Ils sont découpés suivant les couches suivantes :

- La couche **model** qui contient les modèles des objets métiers (Entity).
- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** aux autres **API**, consomme et produit des **DTO** sous forme de **JSON** (**JavaScript Object Notation**).
- La couche **business** qui fait la liaison entre le **controller** et la base de données en manipulant les **DTOs** et les **Entitys**.

5.4.2 - Structure des sources

```
├─ <stock-api>
│   ├─ src
│   │   ├─ main
│   │   │   ├─ java
│   │   │   │   └─ com.pizzaflow.stockapi
│   │   │   │       ├─ configuration
│   │   │   │       ├─ controller
│   │   │   │       ├─ dto
```

```
│   │   │   │       ├── exception
│   │   │   │       ├── model
│   │   │   │       ├── repository
│   │   │   │       ├── service
│   │   │   │       └── StockApiApplication.java
│   │   │   └── resources
│   │   │       └── bootstrap.properties
│   │   └── test
│   │       ├── java
│   │       └── ressources
│   │           └── application.properties
│   │
│   ├── .gitignore
│   ├── .gitlab-ci.yml
│   ├── pom.xml
│   └── README.md
```

5.5 - web-api

5.5.1 - Les couches

Les microservices frontend forment la partie visible par les utilisateurs qui présente une interface graphique. Ils communiquent avec les autres microservices en utilisant des requêtes HTTP. Ils sont découpés suivant les couches suivantes :

- La couche **vue** qui constitue l'interface entre l'utilisateur et l'application et permet de saisir et afficher les données des **DTO**.
- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** et permet d'exécuter les opérations déclenchées par la couche **vue**.

5.5.2 - Structure des sources

```
├── <web-api>
│   ├── src
│   │   ├── main
│   │   │   ├── java
│   │   │   │   └── com.pizzaflow.webapi
│   │   │   │       ├── configuration
│   │   │   │       ├── controller
│   │   │   │       ├── dto
│   │   │   │       ├── exception
```

```
│   │   │   │       ├─ proxy
│   │   │   │       ├─ service
│   │   │   │       ├─ pathTable.java
│   │   │   │       └─ WebApiApplication.java
│   │   │   └─ resources
│   │   │       ├─ locale
│   │   │       │   └─ Resource Bundle 'messages'
│   │   │       │       ├─ messages.properties
│   │   │       │       ├─ messages_en.properties
│   │   │       │       └─ messages_fr.properties
│   │   │       ├─ templates
│   │   │       └─ bootstrap.properties
│   │   └─ test
│   │       ├─ java
│   │       └─ ressources
│   │           └─ application.properties
│   │
│   ├─ .gitignore
│   ├─ .gitlab-ci.yml
│   ├─ pom.xml
│   └─ README.md
```

5.6 - production-api

5.6.1 - Les couches

Les microservices frontend forment la partie visible par les utilisateurs qui présente une interface graphique. Ils communiquent avec les autres microservices en utilisant des requêtes HTTP. Ils sont découpés suivant les couches suivantes :

- La couche **vue** qui constitue l'interface entre l'utilisateur et l'application et permet de saisir et afficher les données des **DTO**.
- La couche **DTO** (**Data Transfer Object**) qui contient la définition des modèles transféré entre les différentes API.
- La couche **controller** qui expose des **endpoints** et permet d'exécuter les opérations déclenchées par la couche **vue**.

5.6.2 - Structure des sources

```
├─ <production-api>
│   ├─ src
│   │   ├─ main
│   │   │   ├─ java
```

```
│   │   │   │   └─ com.pizzaflow.productionapi
│   │   │   │       ├─ configuration
│   │   │   │       ├─ controller
│   │   │   │       ├─ dto
│   │   │   │       ├─ exception
│   │   │   │       ├─ proxy
│   │   │   │       ├─ service
│   │   │   │       ├─ pathTable.java
│   │   │   │       └─ ProductionApiApplication.java
│   │   │   └─ resources
│   │   │       ├─ locale
│   │   │       │   └─ Resource Bundle 'messages'
│   │   │       │       ├─ messages.properties
│   │   │       │       ├─ messages_en.properties
│   │   │       │       └─ messages_fr.properties
│   │   │       ├─ templates
│   │   │       └─ bootstrap.properties
│   │   └─ test
│   │       ├─ java
│   │       └─ ressources
│   │           └─ application.properties
│   │
│   ├─ .gitignore
│   ├─ .gitlab-ci.yml
│   ├─ pom.xml
│   └─ README.md
```

5.7 - gestion-api

5.7.1 - Les couches

Contrairement aux autres parties qui sont divisées en deux pour permettre un meilleur contrôle, les microservice de gestion concentre en lui les couches frontend et backend. Il est moins sensible à la sécurité et à la production car il est utilisé uniquement par les Manager et la Direction. Il communique avec les autres microservices en utilisant des requêtes HTTP. Il est découpé suivant les couches suivantes :

- La couche **model** qui contient les modèles des objets métiers (Entity).
- La couche **DTO** qui contient la définition des modèles transféré entre les différentes API et la couche **vue**.
- La couche **business** qui fait la liaison entre le **controller** et la base de données en manipulant les **DTOs** et les **Entitys**.
- La couche **vue** qui constitue l'interface entre l'utilisateur et l'application et permet de saisir et afficher les données des **DTO**.

- La couche **controller** qui expose des **endpoints** et permet d'exécuter les opérations déclenchées par la couche **vue**.

5.7.2 - Structure des sources

```
<gestion-api>
  ├─ src
  │    ├─ main
  │    │    ├─ java
  │    │    │    └─ com.pizzaflow.gestionapi
  │    │    │         ├─ configuration
  │    │    │         ├─ controller
  │    │    │         ├─ dto
  │    │    │         ├─ exception
  │    │    │         ├─ model
  │    │    │         ├─ proxy
  │    │    │         ├─ service
  │    │    │         ├─ pathTable.java
  │    │    │         └─ GestionApiApplication.java
  │    │    └─ resources
  │    │         ├─ locale
  │    │         │    └─ Resource Bundle 'messages'
  │    │         │         ├─ messages.properties
  │    │         │         ├─ messages_en.properties
  │    │         │         └─ messages_fr.properties
  │    │         ├─ templates
  │    │         └─ bootstrap.properties
  │    └─ test
  │         ├─ java
  │         └─ ressources
  │              └─ application.properties
  │
  ├─ .gitignore
  ├─ .gitlab-ci.yml
  ├─ pom.xml
  └─ README.md
```

6 - POINTS PARTICULIERS

6.1 - Gestion des logs

Pour les logs de l'application on utilisera **SLF4J** (**Simple Loggin Facade for Java**) qui propose une API générique rendant la journalisation indépendante de la mise en œuvre réelle. La journalisation **SLF4J** est incluse dans le package de **Spring Boot**. Pour tous les microservices les fichiers log seront écrites dans le répertoire :

- **/var/log/pizzaflow**

Ci-dessous la configuration dans les fichiers **properties** pour les logs.

```
## configuration du pool de connexion par défaut
logging.level.org.springframework.web=ERROR
logging.level.com.pizzaflow=DEBUG

## le pattern pour la console
logging.pattern.console= "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"

## le pattern pour le nom du fichier
logging.pattern.file=  "%d{yyyy-MM-dd  HH:mm:ss}[%thread]%-5level  %logger{36}
- %msg%n"

## le nom du fichier de log
logging.file=/var/log/pizzaflow/[nom-microservice].log
```

On remplace la partie entre crochets par le nom du microservice correspondant.

6.2 - Fichiers de configuration

On utilise un serveur de configuration pour centraliser les propriétés. Chaque microservice aura son fichier **bootstrap.properties** pour savoir où il doit aller chercher son fichier de configuration. Les fichiers properties de chaque microservice seront dans le sous-répertoire **properties-repository** de **config-server** et avant d'être publier dans un repository sur **GitLab**. Chaque microservice aura sa

configuration de test complète dans un fichier **application.properties** dans **tests/resources**.

Pour chaque microservice on aura un fichier de configuration de test et un de production avec les dénominations suivante :

- [nom-microservice]-test.properties
- [nom-microservice]-prod.properties

On remplace la partie entre crochets par le nom du microservice.

6.2.1 - user-api

6.2.1.1 - Fichier properties

```
## port par défaut pour la première instance de user-api
## utiliser la gamme des ports 4XXX pour les duplications d'instances
server.port=4000

## Propriétés générales de l'application PizzaFlow
pizzaflow.nomPropriété=Valeur

## Propriétés particulières du microservice user-api
pizzaflow.user.nomPropriété=Valeur

## Datasource en fonction de l'environnement test ou production

## Properties pour les logs
```

6.2.1.2 - Datasources test

```
## configuration du pool de connexion par défaut
spring.datasource.hikari.connectionTimeout=20000
spring.datasource.hikari.maximumPoolSize=5
spring.datasource.initialize=true
```

```
## configuration de H2
spring.datasource.url=jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-
1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.driverClassName=org.h2.Driver

## identifiants temporaire de connexion à changer et crypter avec Jasypt pour
la production
spring.datasource.username=PizzaFlow
spring.datasource.password= PizzaFlow

spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
```

6.2.1.3 - Datasources production

```
## configuration du pool de connexion par défaut
spring.datasource.hikari.connectionTimeout=20000
spring.datasource.hikari.maximumPoolSize=5

## configuration de PostgreSQL remplacer localhost par la vrai adresse IP
spring.datasource.url=jdbc:postgresql://localhost:5432/user
spring.datasource.driverClassName=org.postgresql.Driver

## identifiants temporaire de connexion à changer et crypter avec Jasypt pour
la production
spring.datasource.username=PizzaFlow_user
spring.datasource.password= PizzaFlow_user

# DANGER!! mettre à create pour refaire le schéma
spring.jpa.hibernate.ddl-auto=update
```

6.2.2 - stock-api

6.2.2.1 - Fichier properties

```
## port par défaut pour la première instance de stock-api
## utiliser la gamme des ports 5XXX pour les duplications d'instances
server.port=5000

## Propriétés générales de l'application PizzaFlow
pizzaflow.nomPropriété=Valeur

## Propriétés particulières du microservice stock-api
pizzaflow.stock.nomPropriété=Valeur

## Datasource en fonction de l'environnement test ou production

## Properties pour les logs
```

6.2.2.2 - Datasources test

```
## configuration du pool de connexion par défaut
spring.datasource.hikari.connectionTimeout=20000
spring.datasource.hikari.maximumPoolSize=5
spring.datasource.initialize=true

## configuration de H2
spring.datasource.url=jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-
1;DB_CLOSE_ON_EXIT=FALSE
```

```
spring.datasource.driverClassName=org.h2.Driver

## identifiants temporaire de connexion à changer et crypter avec Jasypt pour la production
spring.datasource.username=PizzaFlow
spring.datasource.password= PizzaFlow

spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
```

6.2.2.3 - Datasources production

```
## configuration du pool de connexion par défaut
spring.datasource.hikari.connectionTimeout=20000
spring.datasource.hikari.maximumPoolSize=5

## configuration de PostgreSQL remplacer localhost par la vrai adresse IP
spring.datasource.url=jdbc:postgresql://localhost:5432/stock
spring.datasource.driverClassName=org.postgresql.Driver

## identifiants temporaire de connexion à changer et crypter avec Jasypt pour la production
spring.datasource.username=PizzaFlow_stock
spring.datasource.password= PizzaFlow_stock

# DANGER!! mettre à create pour refaire le schéma
spring.jpa.hibernate.ddl-auto=update
```

6.2.3 - web-api

6.2.3.1 - Fichier properties

```
## port par défaut pour la première instance de web-api
## utiliser la gamme des ports 7XXX pour les duplications d'instances
server.port=7000

## Propriétés générales de l'application PizzaFlow
pizzaflow.nomPropriété=Valeur

## Propriétés particulières du microservice web-api
pizzaflow.web.nomPropriété=Valeur

## Properties pour les logs
```

6.2.4 - production-api

6.2.4.1 - Fichier properties

```
## port par défaut pour la première instance de production-api
## utiliser la gamme des ports 8XXX pour les duplications d'instances
server.port=8000

## Propriétés générales de l'application PizzaFlow
pizzaflow.nomPropriété=Valeur

## Propriétés particulières du microservice production-api
pizzaflow.production.nomPropriété=Valeur
```

6.2.5 - gestion-api

6.2.5.1 - Fichier properties

```
## port par défaut pour la première instance de gestion-api
## utiliser la gamme des ports 9XXX pour les duplications d'instances
server.port=9000

## Propriétés générales de l'application PizzaFlow
pizzaflow.nomPropriété=Valeur

## Propriétés particulières du microservice gestion-api
pizzaflow.gestion.nomPropriété=Valeur

## Datasource en fonction de l'environnement test ou production

## Properties pour les logs
```

6.2.5.2 - Datasources test

```
## configuration du pool de connexion par défaut
spring.datasource.hikari.connectionTimeout=20000
spring.datasource.hikari.maximumPoolSize=5
spring.datasource.initialize=true

## configuration de H2
spring.datasource.url=jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-
1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.driverClassName=org.h2.Driver

## identifiants temporaire de connexion à changer et crypter avec Jasypt pour
la production
spring.datasource.username=PizzaFlow
```

```
spring.datasource.password= PizzaFlow

spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
```

6.2.5.3 - Datasources production

```
## configuration du pool de connexion par défaut
spring.datasource.hikari.connectionTimeout=20000
spring.datasource.hikari.maximumPoolSize=5

## configuration de PostgreSQL remplacer localhost par la vrai adresse IP
spring.datasource.url=jdbc:postgresql://localhost:5432/gestion
spring.datasource.driverClassName=org.postgresql.Driver

## identifiants temporaire de connexion à changer et crypter avec Jasypt pour
la production
spring.datasource.username=PizzaFlow_gestion
spring.datasource.password= PizzaFlow_gestion

# DANGER!! mettre à create pour refaire le schéma
spring.jpa.hibernate.ddl-auto=update
```

6.3 - Ressources

6.3.1 - Graphiques

Les codes graphiques, les choix de polices, de logos et de photos des produits vendus seront fournis par **OC Pizza**.

6.3.2 - Données

OC Pizza fournira les données de base nécessaires au fonctionnement de l'application comme :

- Les produits utilisés, leur composition, conditionnement, prix et références fournisseur.
- Les recettes des produits facturés avec les ingrédients, les proportions leur prix de vente et taxes.
- Les références des magasins et des employés pour la création des comptes utilisateurs.
- Les références nécessaires au paiement bancaire.

6.4 - Environnement de développement

Le choix de l'environnement de développement reste libre à la charge des développeurs. Toutefois, **IntelliJ IDEA** version 2020.1 intègre de nombreux plugins pour faciliter la production de code en J2EE avec :

- **Git Flow Integration** 0.7.2 de Opher Vishnia pour l'utilisation de **Git Flow**.
- **GitLab Project** 2.0.1 de Pavel Polivka pour l'intégration à **GitLab**.
- **Lombok** 0.30.2020.1 de Michail Plushnikov pour l'utilisation de l'annotation **@Data**.
- **SonarLint** 4.7.0.17141 de SonarSource pour la vérification du code.
- **AWS Toolkit** pour une intégration de **AWS CLI**.
- **Maven**, **Markdown**, **Properties**, **Resource Bundle Editor**, **YAML**, **Git**.

6.5 - Procédure de packaging / livraison

L'application sera déployée sur **AWS** lors de la livraison finale. Un dossier d'exploitation sera remis au même instant pour indiquer les procédures d'installation, de démarrage, d'arrête et de mise à jour de l'application.

7 - GLOSSAIRE

AMI	(**Amazon Machine Image**) logiciel d'exploitation Amazone de type linux.
AWS	(**Amazon Web Services**) service internet d'Amazon.
CNIL	(**Commission nationale de l'informatique et des libertés**).
CSS	(**Cascading Style Sheets**) fichier de style pour la présentation des pages **HTML**.
DTO	(**Data Transfer Object**) type d'objet permettant de transférer des données.
EC2	(**Elastic Cloud Compute**) serveur de base permettant d'intégrer de nombreux systèmes.
IAM	(**Identity and Access Management**) service d'Amazon pour gérer l'authentification des utilisateurs.
Jasypt	(**Java Simplified encryption**) librairie java qui permet d'effectuer un cryptage basic dans des fichiers de configuration.
JDK	(**Java Developer Kit**) outils de développement du langage **Java**.
JRE	(**Java Runtime Environment**) outils pour exécuter un exécutable **Java**.
JS	**Javascript** est un langage de script pour les pages web.
JSON	(**JavaScript Object Notation**) format léger d'échange de données facilement compréhensible par l'homme et manipulable par l'ordinateur.
load-balancing	Action de répartir la charge entre plusieurs instances d'une même application.
NoSQL	(**Not Only SQL**) Type de base de données qui n'utilise pas l'architecture classique des bases de données relationnelles **SQL**.
RDS	(**Relational Database Service**) Serveur avec un **SGBD-R** intégré.
Repository	Répertoire dans le cloud.
S3	(**Simple storage Service**) serveur de fichiers static pour les sites web.
SGBD-R	(**Système de Gestion de Bases de Données Relationnelles**).
SLF4J	(**Simple Loggin Facade for Java**) couche abstraite pour l'utilisation de différents loggers.