OC Pizza

Système de gestion pour pizzeria

Dossier de conception technique

Version 1.2

Auteur
Brice NIATEL
Développeur d'application android

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
Brice NIATEL	28/07/2021	Création du document	1.0
Brice NIATEL	18/08/2021	Finition de la rédaction du document	1.1
Brice NIATEL	18/08/2021	Relecture	1.2

2 - INTRODUCTION

2.1 - Objet du document

Le présent document constitue le dossier de conception technique de l'application ***OC Pizza*** pour les développeurs, à l'équipe technique et à la maintenance.

L'objectif du document est de décrire les solutions techniques du système informatique (développement et maintenance) afin de répondre aux besoins exprimés par le groupe ***OC Pizza***.

Les éléments du présent dossier découlent :

- Des besoins clients transmis au projet 6,
- Des spécifications fonctionnelles du projet 6.

2.2 - Références

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

1. **Projet 10 - Dossier de conception fonctionnelle :** Dossier de conception fonctionnelle de l'application,
2. **Projet 10 - Dossier d'exploitation :** Dossier d'exploitation de l'application,
3. **Projet 10 - Dossier de conception technique :** Dossier de conception d'exploitation de l'application,
4. **Projet 10 - PV Livraison :** Procès-verbal de la livraison finale.

3 - Architecture Technique

3.1 - Composants généraux

La modélisation et l'identification des éléments composant le système d'information à mettre en place et leurs interactions, est illustré par le digramme de composants en figure 1.

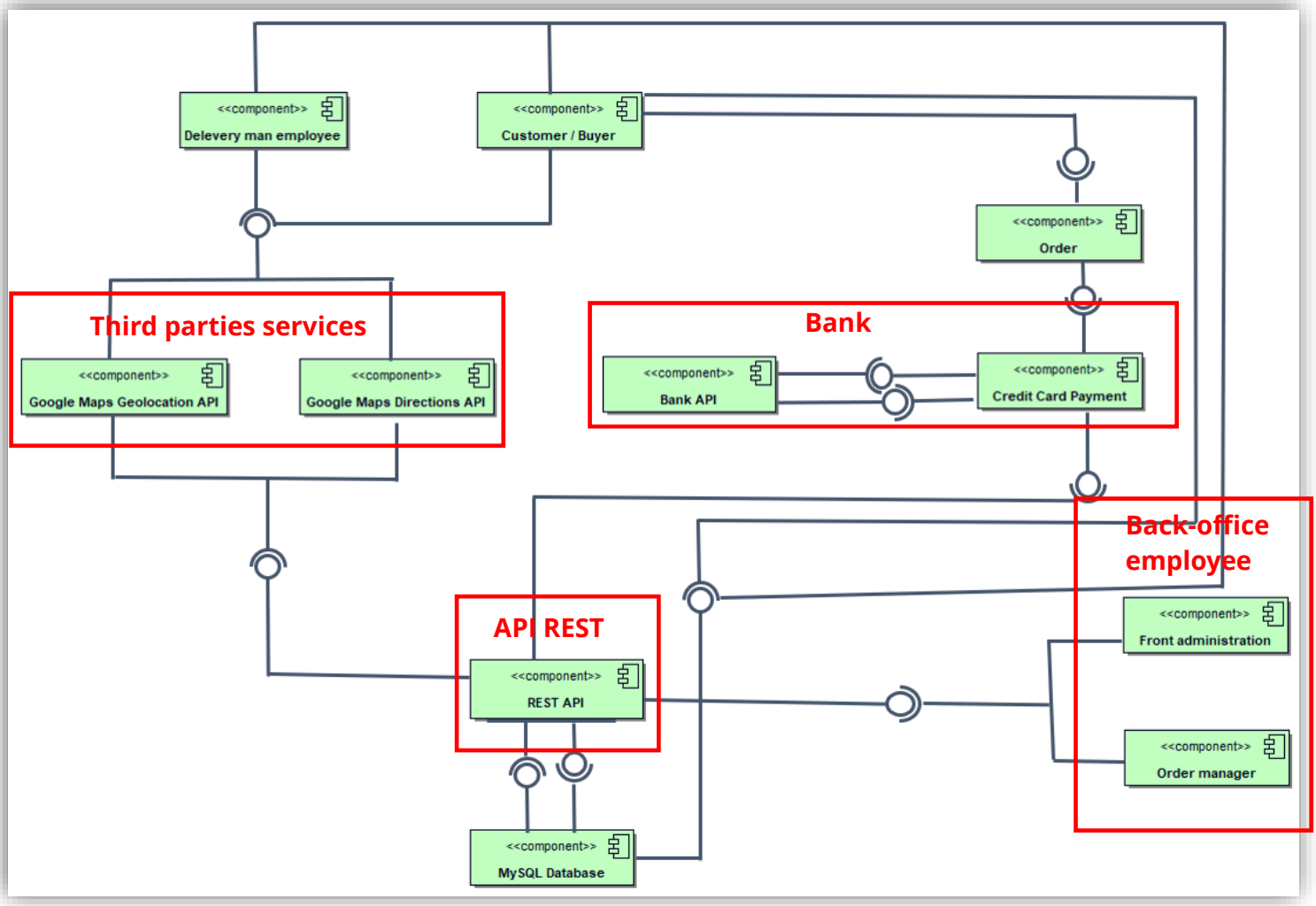

Figure 1 : Diagramme de composants

3.1.1 - Package Third parties services

Ce package regroupe les différentes API utilisées par le système.

3.1.1.1 - Google Maps Geolocation API

L'API de géolocalisation renvoie un rayon de localisation et de précision basé sur des informations sur les tours de téléphonie cellulaire et les nœuds Wifi que le client mobile peut détecter. Ce document décrit le protocole utilisé pour envoyer ces données au serveur et pour renvoyer une réponse au client.

Celle-ci a pour rôle de détecter les restaurants proches de l'utilisateur ainsi que sa localisation pour les livraisons, de plus les livreurs ont besoin de savoir où le client se situe.

3.1.1.2 - Google Maps Directions API

L'API Directions est un service Web qui utilise une requête pour renvoyer des directions à l'appareil entre les emplacements. Vous pouvez recevoir des itinéraires pour plusieurs modes de transport, tels que les transports en commun, la voiture, la marche ou le vélo.

Celle-ci permettra l'acheminement de la commande du client par le livreur.

3.1.2 - Package Back office employee

Ce package regroupe la gestion du restaurant.

3.1.2.1 - Front administration

Interface du site web dédié aux employés pour la gestion des commandes, de statistiques, la gestion du restaurant et les livreurs.

3.1.2.2 - Order manager

Gérer toutes les commandes qui ont été passés selon les clients et répercuter les actions dans le système.

3.1.3 - Package BANK

Ce package regroupe la gestion globale de paiement de l'application.

3.1.3.1 - Bank API

L'API de paiement en ligne est un service qui permet de transmettre directement les informations de paiement à l'application tel que la vérification de conformité des informations et le paiement.

3.1.3.2 - Credit card payment

Le composant ***Credit card payment*** permet de récupérer les données de la commande tel que le prix et également les informations du client tel comme son nom et prénom par

exemple. Ce composant va ensuite échanger avec le système bancaire **Bank API**.

3.1.4 - API Rest

Une API REST est une interface de programmation d'application (API ou API web) qui respecte les contraintes du style d'architecture REST et permet d'interagir avec les services web REST.

Une API est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications.

3.2 - Application

Le serveur d'application sera Apache TomCat couplé avec Java EE.

3.2.1 - Composants Base de données

La base de données est sur un serveur MySQL qui est lié au back end de l'application. Son rôle est de stocker les données utilisateurs ainsi que celles de l'application elle-même tel que les pizzas, commandes, logs…

3.2.2 - Composants Front End – Angular

Le front end est l'interface qu'aura l'utilisateur, qu'il soit client ou personnel de commande, chaque rôle aura une interface qui sera adaptée. Une personne travaillant au sein du restaurant aura plus de fonctionnalités qu'un client.

Angular permet de répondre au besoin, on utilisera le framework Ionic qui permettra de compiler pour les applications Android, iOs et Web, dans le but de maintenir un seul code pour les trois plateformes.

3.2.3 - Composants Back end – Java

Le back end permet de traiter les actions utilisateurs et faire fonctionner l'application avec toute sa structure et ses modules. Il va faire des demandes et renvoyer les informations à l'utilisateur via l'affichage (front end).

Java EE permet de répondre à ce besoin sous un serveur TomCat.

3.3 - Application Mobile

L'application mobile sera une application web tout en étant responsive pour être adaptée à tout affichage. Ceci permettra un gain de temps et d'argent tout en étant aussi efficace pour les besoins du projet.

4 - Architecture de Deploiement

L'architecture de déploiement est l'ensemble du système logiciel et matériel.

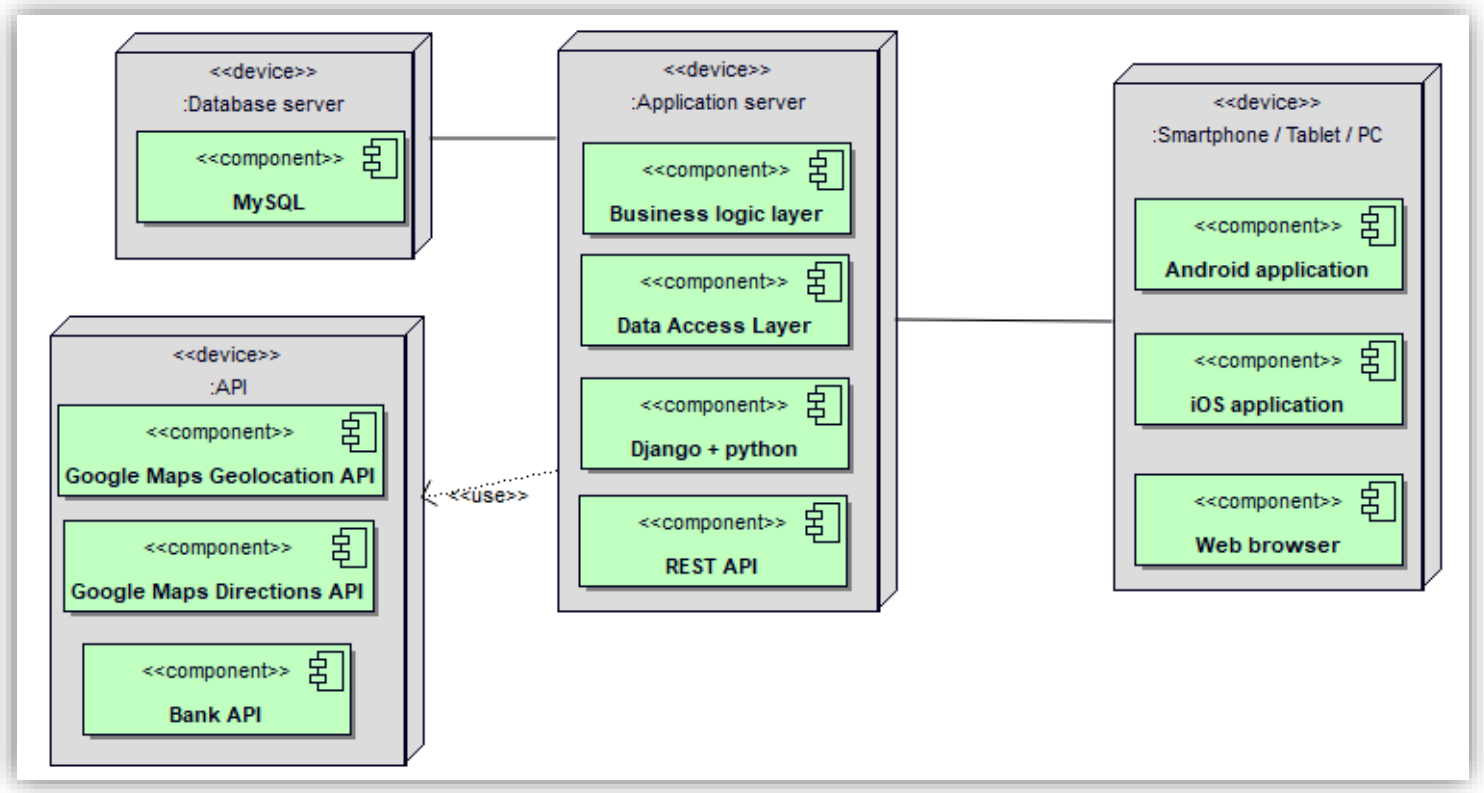

Figure 2 : Diagramme de déploiement

4.1 - Base de données

Les données sont stockées sur une base de données ***PostreSQL*** hébergé en ligne sur Clever Cloud. En figure 3 voici le modèle physique de base de données prévu pour la solution technique.

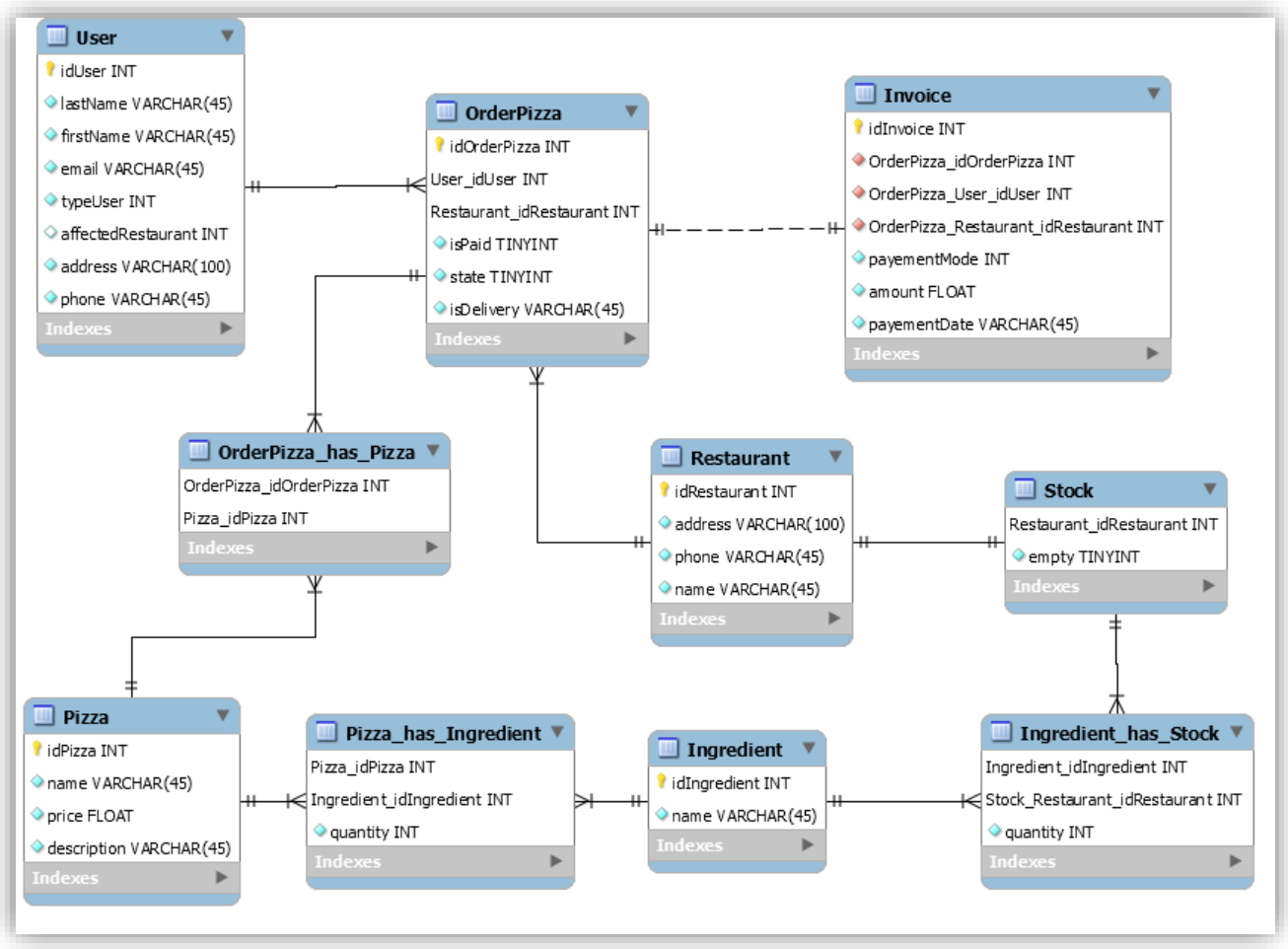

Figure 3 : Modèle physique de base de données

4.2 - Application Back-Office

Le back-office sera déployé sur une autre instance de Clever Clever cloud.

5 - ARCHITECTURE LOGICIELLE

5.1 - Principes généraux

L'hébergement et le management des versions du projet seront gérées sur ***GitHub***. De plus, les dépendances ainsi que le packaging se feront via ***Apache Maven***. Pour manager la persistance des données en base de données MySQL nous utiliserons ***Clever Cloud*** avec son service MySQL Hosting.

La solution s'articule autour de plusieurs modules autonomes exposants des ***API Rest*** utilisables par les autres modules.

Les modules sont documentés et exposés via **Swagger** sur une interface HTML accessibles aux développeurs permettant ainsi de faciliter le développement.

5.1.1 - Les couches

L'architecture applicative de chaque module est structurée selon le concept ***Domain Driven Design*** :

- **Application** (API) : responsable de la parie opérationnelle du module. Permet d'exposer le métier et le model sans jamais entrer dans le business, de faire la transition entre les couches applicatives (interne/externe). Cette partie contient également le paramétrage et les fonctions nécessaires au lancement du service.
- **Model** : implémentation du modèle des objets métiers. Gère le traitement et la logique métier des objets manipulés.
- **Infrastructure** : responsable de la partie des traitements en BDD. Elle répond aux demandes de la couche model et assure la persistance en retour des données traitées.

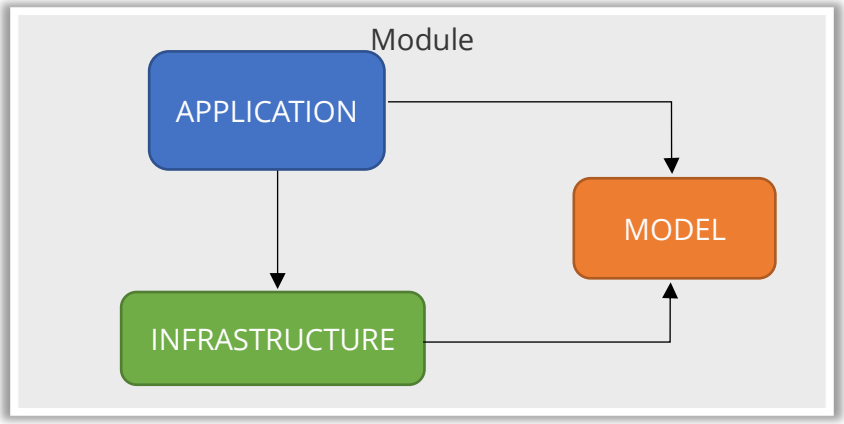

Figure 4 : Structure d'un module

5.1.2 - Les modules

Un module est un fichier de code de programmation ou un fichier de bibliothèque, il vit indépendamment de l'application et est consommée par celle-ci.

5.1.2.1 - Module config

Paramétrage de l'application avec des fichiers de configuration externes sur un cloud. Celui-ci facilite le paramétrage (évite une livraison supplémentaire) de l'application et les différents modules.

5.1.2.2 - Module pizzaWebApp

Affichage sur les appareils de l'application.

5.1.2.3 - Module gateway

Gestion des demandes de l'interface utilisateur, ce module interroge les différents modules concernés dans le but de répondre aux demandes du module pizzaWebApp.

5.1.2.4 - Module user

Gestion des données utilisateurs tel que client et personnel de restaurant.

5.1.2.5 - Module stock

Gestion des différents produits et des stocks.

5.1.2.6 - Module order

Gestion des commandes jusqu'à la livraison.

5.1.2.7 - Module payment

Gestion des transactions bancaires sur l'application et dans les restaurants.

5.1.2.8 - Module task-manager

Gestion de la planification des tâches périodiques tel que les différentes sauvegardes.

5.1.3 - Structure des sources

Les répertoires du projet sont structurés de la manière suivante :

- Le répertoire racine contient le fichier POM parents de module.
- Les modules sont structurés suivant le concept ***Domain Driven Design***.


```
racine
├─ pom.xml
├─ <config>
│   ├─ pom.xml
│   └─ src
│       ├─ main
│       │   ├─ java
│       │   │   └─ com.oc-pizza.www
│       │   │           └─ CloudConfigApp.java
│       │   └─ resources
│       │       └─ application.properties
│       └─ test
│           ├─ java
│           └─ resources
├─ <user>
│   ├─ pom.xml
│   └─ src
│       ├─ main
│       │   ├─ java
│       │   │   └─ com.oc-pizza.www
│       │   │           ├─ api/
│       │   │           ├─ model/
│       │   │           └─ infrastructure/
│       │   └─ resources
│       │       └─ application.properties
│       └─ test
│           ├─ java
│           └─ resources
├─ <pizzaWebApp>
│   ├─ pom.xml
│   ├─ package.json
│   ├─ README.md
│   └─ src
│       ├─ components/
│       ├─ css/
│       ├─ lib/
│       ├─ services/
│       ├─ utils/
│       ├─ app.js
│       ├─ route.js
│       ├─ config.js
│       └─ shared-modules.js
└─ ui
    └─ assets
        ├─ boostrap/
        ├─ fontawesome/
        ├─ jquery/
        ├─ params/
```

Les autres modules : ***gateway***, ***stock***, ***order***, ***payment***, ***task-manager*** ont une structure similaire à celle du module ***user*** qui est détaillé ci-dessus.

6 - POINTS PARTICULIERS

6.1 - Gestion des logs

Les logs sont sauvegardés tous les jours et conservés pour une durée d'au minimum un mois sur le serveur pour chacun des modules.

6.2 - Fichiers de configuration

6.2.1 - Application OC Pizza

Le fichier de configuration est accessible le module config pour chaque module, celui-ci se nomme « *MODULE.properties* ». De plus le module possède aussi un fichier de configuration « *bootstrap.properties* » permettant de définir les différent paramétrages.

6.2.1.1 - Fichier module.properties

Les sources de donnée sont paramétrées dans le « *fichier.properties* » de config. Ce fichier contient l'ensemble des paramètres du module qui ne sont pas nécessaire au lancement et au chargement du contexte.

6.2.1.2 - Fichier bootstrap.properties

Contient les informations relatives au fonctionnement du module tel que le nom et le port pour l'enregistrement auprès d'Eureka Server.

6.3 - Ressources

Linux CentOS : https://www.centos.org/

Maven Repository : https://mvnrepository.com/

Apache Maven : https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Spring initializr : https://start.spring.io/

Angular Material : https://material.angular.io/components/categories

Bootstrap : https://getbootstrap.com/docs/4.5/getting-started/introduction/

Angular CLI : https://angular.io/cli

Clever cloud : https://www.clever-cloud.com/en/

Swagger : https://swagger.io/

MySQL : https://www.mysql.com/fr/downloads/

MariaDB : https://mariadb.org/documentation/

6.4 - Environnement de développement

Nous utiliserons pour le développement des applications mobiles, ***Android Studio*** pour l'application android web, ***Xcode*** pour l'application iOS web et ***Webstorm*** pour l'application web (Angular / Bootstrap).

Pour le développement back-office en Java EE nous utiliserons ***Eclipse*** avec les modules Springs.

L'accès aux bases de données se fera par l'intermédiaire d'un logiciel d'administration de BDD tel que ***MariaDB***.

6.5 - Procédure de packaging / livraison

Le packaging est fait par Maven et celui-ci est configuré dans Spring au niveau du traitement de build. Ci-dessous le processus de Maven :

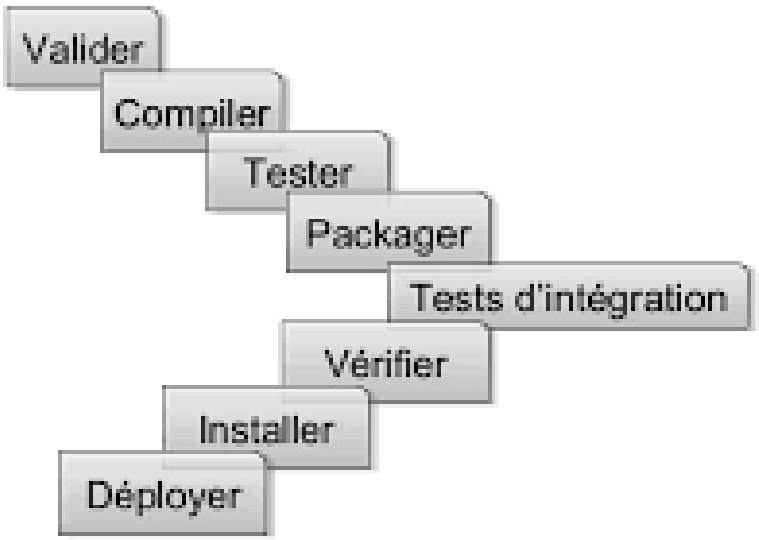

Figure 5 : Cycle de vie Maven

La procédure de packaging est alors lancée après toutes les étapes précédentes, la commande utilisée sera « ***mvn clean package*** », il en résultera un fichier exécutable java « ***.jar*** ».

Le déploiement se fera avec la commande « ***mvn clean install*** » qui permettra de supprimer les anciennes compilations de classe java ainsi que les fichiers ressources afin de compiler, tester, packager et installer sur un environnement propre.

Les livraisons se feront toutes les deux semaines à chaque fin de sprint, ce seront des livraisons mineures tel que des corrections de bogues remontés par le client / utilisateurs. Les livraisons majeures se feront toutes les quatre semaines tel que des ajouts de fonctionnalités ou correctif de bogue majeur.

7 - GLOSSAIRE

Domain Driven Design : Référence à la conception pilotée par le métier. C'est une approche de développement de logiciels centrée sur le métier au travers de design patterns de conception (technique), des modèles conceptuels.

Spring : C'est un Framework contient de nombreuses fonctionnalités, qui sont bien organisées en une vingtaine de modules. Ces modules peuvent être regroupés en fonction de leurs fonctionnalités principales dans Core Container, Data Access / Integration, Web, AOP (Aspect Oriented Programming), Instrumentation et Test.

REST : Signifie REpresentational State Transfer, constitue un style architectural et un mode de communication fréquemment utilisé dans le développement de services Web.