OC Pizza

Système de gestion de Pizzerias

Dossier de conception technique

Version 1.3

Auteur
Rémy VALLET
Développeur d'application

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
Rémy VALLET	23/10/2020	Création du document	1.0
Rémy VALLET	30/10/2020	Rédaction des chapitres 2, 3 et 4	1.1
Rémy VALLET	06/11/2020	Rédaction des chapitres 5 et 6	1.2
Rémy VALLET	13/11/2020	Révision et finalisation chapitre 6 et 7	1.3

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, à la maintenance applicative et à l'équipe technique.

L'objectif de la conception technique est de préciser les règles de gestion applicatives pour le développement et la maintenance du système d'information.

Les éléments concernés sont :

- L'architecture technique des composants généraux des différents packages.
- Les différents composants de l'application.
- L'architecture de déploiement de la solution.
- L'architecture logicielle.
- Les points particuliers mis en œuvre pour cette solution tel que les fichiers de configurations, les logs, les ressources, l'environnement ou la procédure de livraison.

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. **Projet_8-PV_Livraison** : Procès-verbal de la livraison finale.

3 - ARCHITECTURE TECHNIQUE

3.1 - Composants généraux

La modélisation et l'identifier des éléments composant le système d'information à mettre en place et leur interaction est illustré par le digramme de composants en figure 1.

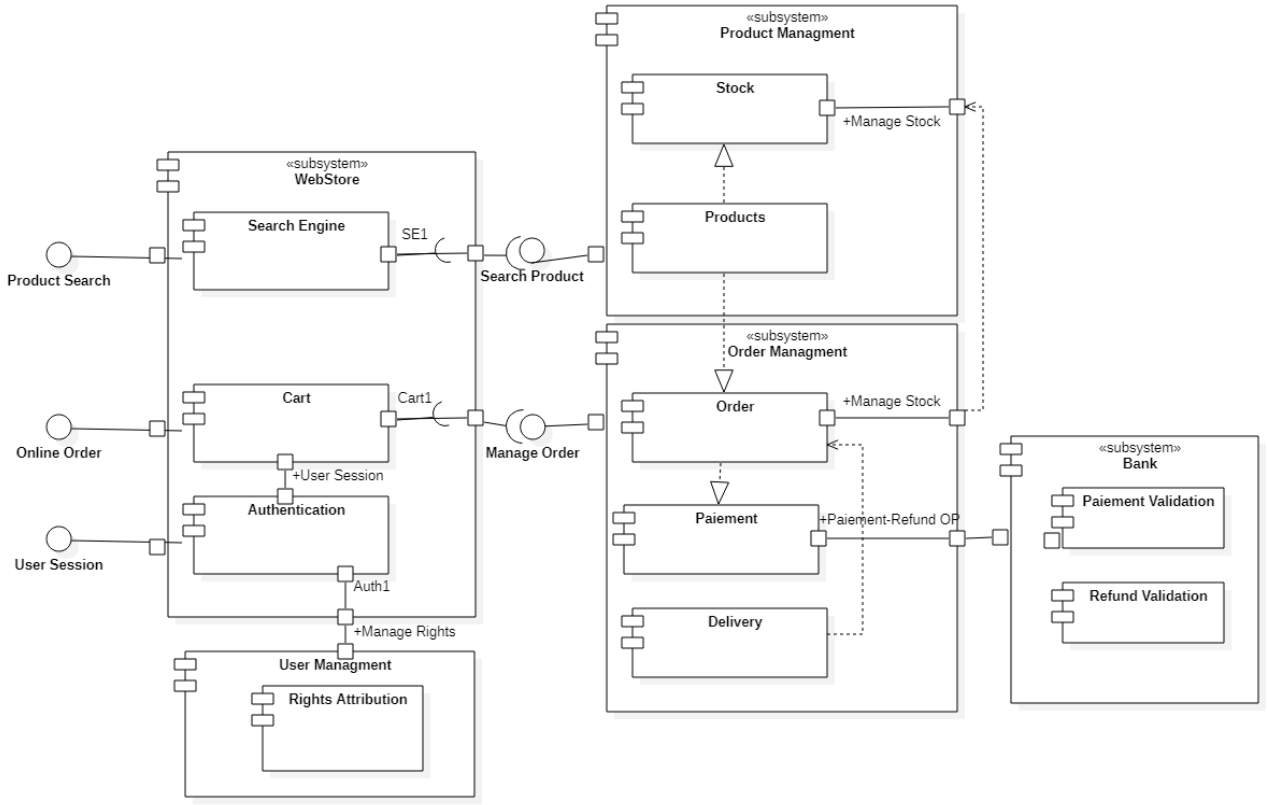


Figure 1: Diagramme de composants

3.1.1 - Package WebStore

Ce package regroupe les fonctionnalités nécessaires à la navigation sur l'application et à la prise de commande.

3.1.1.1 - Composant Search Engine

Ce composant permet l'échange et le tri de données entre l'application et la base de données. Une API Rest qui attend en entrée des paramètres de requêtes définis et fournis une réponse paginé et trié selon les paramètres reçus. Les requêtes peuvent provenir d'un formulaire exposé à l'utilisateur de l'application (client, employé) ou d'un applicatif tiers (affichage du stock d'un produit, d'une liste de commande, d'une recette…)

3.1.1.2 - Composant Cart

Ce composant permet la création d'une commande par un utilisateur. Selon le profil de la session utilisateurs il expose des options plus ou moins complètes, notamment la possibilité de passer une commande.

3.1.1.3 - Composant Authentification

Ce composant permet l'authentification de l'utilisateur et la création de la session. Il interagit avec le package User Management pour l'attribution des droits de la session utilisateur.

3.1.2 - Package User Management

3.1.2.1 - Composant Rights Attribution

Ce composant permet de remonter les droits utilisateurs sur la session lors de l'authentification.

3.1.3 - Package Product Management

3.1.3.1 - Composant Stocks

Ce composant permet de mettre à jour les mouvements de stock et de répondre aux requêtes d'information de stock, notamment pour l'affichage des produits, l'affichage des recettes et le calcul de disponibilité des produits au panier lors de la validation de commande.

3.1.3.2 - Composant Products

Ce composant permet l'affichage des produits. Il contient les informations détaillé des produits, notamment la constitution des produits composés. Il interagit avec le composant stock pour remonter le stock disponible.

3.1.4 - Package Order Management

3.1.4.1 - Composant Order

Ce composants permet l'affichage des commandes et contient les informations relatives aux commandes, notamment les lignes de commandes, la validation et l'annulation de commande. Il interagit avec le composant stock pour envoyer les mouvements de stocks suite à une validation ou à une annulation de commande. Il est en charge de l'envoie d'une demande de paiement ou de remboursement.

3.1.4.2 - Composant Paiement

Ce composant permet les traitements relatifs aux transactions bancaires de validation de paiement et de remboursement.

3.1.4.3 - Composant Delivery

Ce composant permet les traitements relatifs à la livraison. Il remonte les informations à la partie commande.

3.1.5 - Package Bank

3.1.5.1 - Composant Paiement Validation

Ce composant intègre l'API bancaire pour les demandes et réponse relatives aux transactions de paiement.

3.1.5.2 - Composant Refund Validation

Ce composant intègre l'API bancaire pour les demandes et réponse relatives aux transactions de remboursement.

4 - ARCHITECTURE DE DEPLOIEMENT

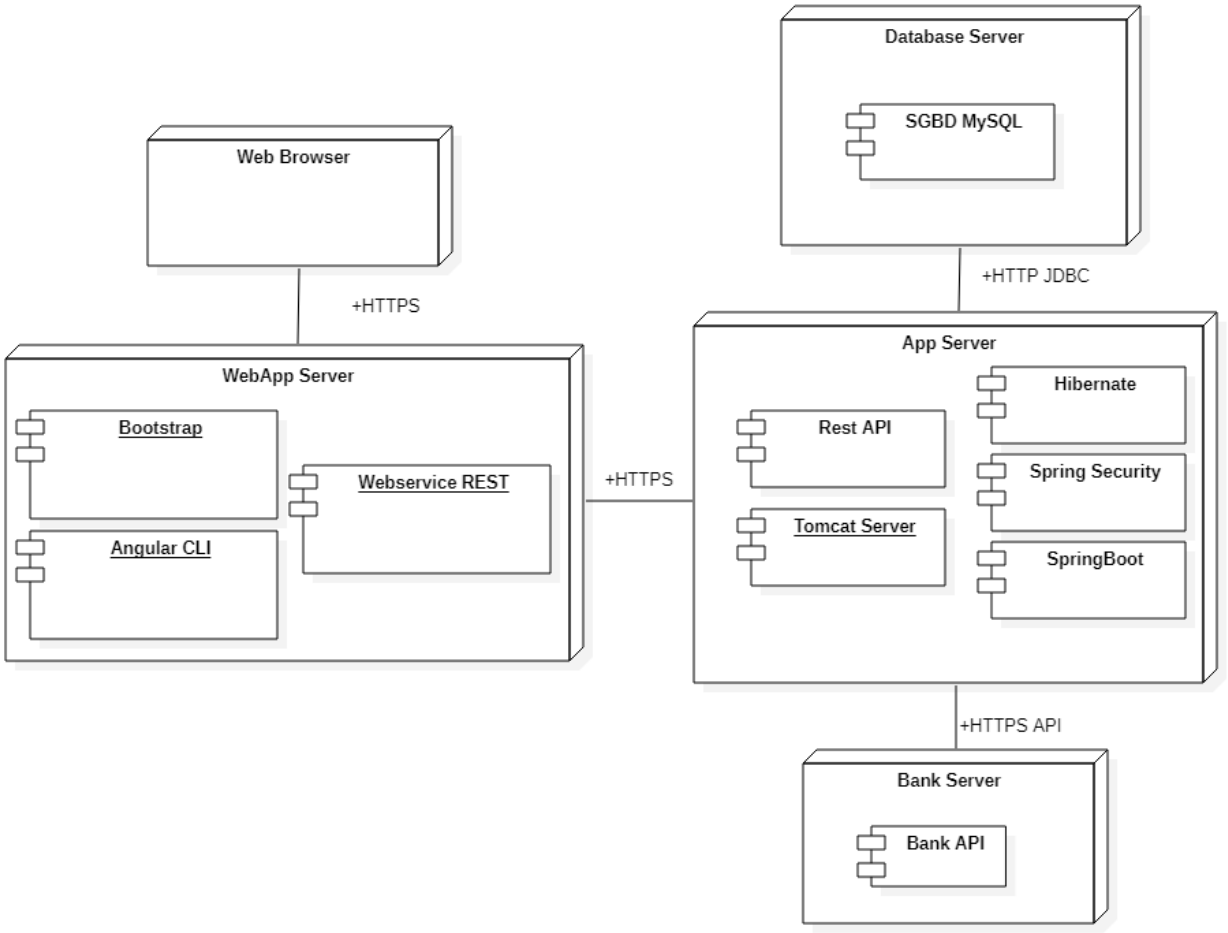


Figure 2: Diagramme de déploiement

TODO : Explication / commentaires si nécessaires...

4.1 - Serveur de Base de données

Les données sont persistées sur une base de données relationnelle SQL sur un serveur MySQL installé sur un système d'exploitation Linux centOS.

Nous retrouvons en figure 3 le modèle physique de données nécessaire à la mise en place de la solution.

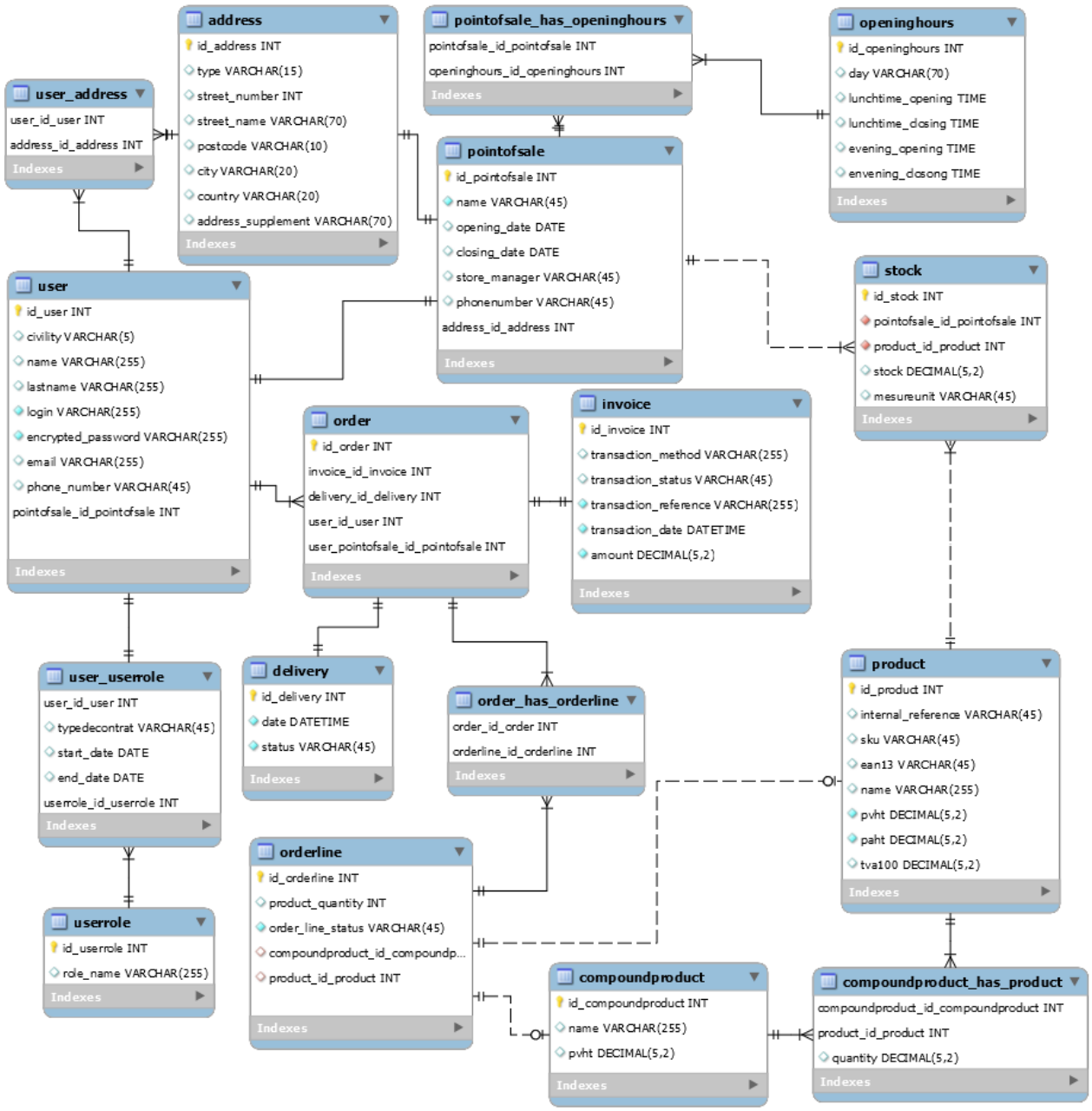


Figure 3 : Modèle Physique de Données (MPD)

4.2 - Serveur d'application Back-Office (BO) et Front-Middle-Office (FMO)

Le BO sera déployé sur un serveur apache Tomcat sous Linux centOS avec le FMO sous Angular CLI.

5 - ARCHITECTURE LOGICIELLE

5.1 - Principes généraux

Les sources et versions du projet sont gérées par **GitLab.** Les dépendances et le packaging par **Apache Maven.** La persistance des données en BDD MySQL par **Hibernate**.

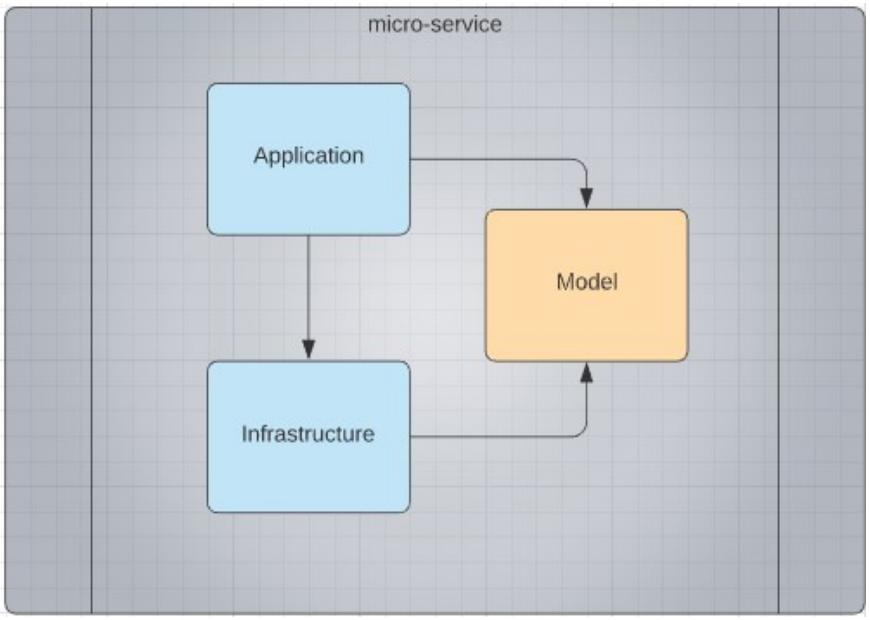
La solution s'articule autour de plusieurs modules de micro-services autonomes exposants des **API Rest** consommables par les autres modules.

Les micro services sont également documentés et exposés via **Swagger** sur une interface HTML accessibles aux développeurs.

5.1.1 - Les couches

L'architecture applicative de chaque micro-service est structuré selon le concept DDD (Domain Driven Design) :

- une couche **application** (API): responsable de la parie opérationnelle du micro-service. 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.
- une couche **model** : implémentation du modèle des objets métiers. Gère le traitement et la logique métier des objets manipulés.
- une couche **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.



5.1.2 - Les modules

5.1.2.1 - Module cloud-config

Ce micro-service est destiné au paramétrage de l'application avec des fichiers de configuration externalisés. Il permet la gestion des configurations de l'application sans avoir à relivrer l'applicatif.

5.1.2.2 - Module pizzaWebApp

Ce module contient l'interface utilisateur, il est en charge de l'affichage de l'application sur les différents devices.

5.1.2.3 - Module gateway-bmo

Ce micro-service est en charge de la réception des demandes de l'interface utilisateurs (IHM). Il joue le rôle de passerelle pour construire une réponse en interrogeant les différents micro-services concernés. Il renvoi la réponse à pizzaWebApp.

5.1.2.4 - Module ms-user

Ce micro-service est en charge des données utilisateurs, notamment les utilisateurs, leurs rôles et leurs adresses.

5.1.2.5 - Module ms-orga

Ce micro-service est en charge de données liées à l'organisation OC Pizza, notamment les points de ventes et les horaires d'ouvertures.

5.1.2.6 - Module ms-product

Ce micro-service est en charge des données liées aux produits, notamment les produits composés et les stocks.

5.1.2.7 - Module ms-order

Ce micro-service est en charge des données liées aux commandes, notamment les statuts, les lignes de commandes et les livraisons.

5.1.2.8 - Module ms-transaction

Ce micro-service est en charge des données liées aux moyens de paiement, notamment à l'encaissement sur les Points de vente et aux transactions bancaires.

5.1.2.9 - Module orchestrator-job

Ce micro-service est destiné à la planification des tâches périodiques (Batch). Il génère les informations de statistiques des différents points de ventes.

5.1.3 - Structure des sources

La structuration des répertoires du projet suit la logique suivante :

- Le répertoire racine contient le fichier POM parents de Maven pour la gestion des dépendances et des versions des libraires tierces et de chaque micro-service.

- Les micro-services sont structuré suivant le concept DDD, nous représenterons uniquement le ms-product ci-après.

```
racine
 ├─ pom.xml
 ├─ <cloud-config>
 │   ├─ pom.xml
 │   └─ src
 │       ├─ main
 │       │   ├─ java
 │       │   │   └─ com.oc-pizza.www
 │       │   │           └─ CloudConfigApp.java
 │       │   └─ resources
 │       │       └─ application.properties
 │       └─ test
 │           ├─ java
 │           └─ resources
 ├─ <ms-product>
 │   ├─ pom.xml
 │   └─ src
 │       ├─ main
 │       │   ├─ java
 │       │   │   └─ com.oc-pizza.www
 │       │   │           ├─ api/
 │       │   │           ├─ model/
 │       │   │           └─ infra/
 │       │   └─ resources
 │       │       └─ application.properties
 │       └─ test
 │           ├─ java
 │           └─ resources
 │
/…Les autres ms conservent une structures similaire à ms-produt…/
 ├─ <orchestrator-job>
 │   ├─ pom.xml
 │   └─ src
 │       ├─ main
 │       │   ├─ java
 │       │   │   └─ com.oc-pizza.www
 │       │   │           ├─ api/
 │       │   │           ├─ model/
 │       │   │           └─ infra/
 │       │   └─ resources
 │       │       └─ application.properties
 │       └─ test
 │           ├─ java
 │           └─ resources
 ├─ <pizzaWebApp>
 │   ├─ pom.xml
 │   ├─ package.json
 │   ├─ README.md
```

```
│   ├── e2e/
│   ├── node_modules/
│   └── src
│       ├── components/
│       ├── css/
│       ├── services/
│       ├── utils/
│       ├── app.js
│       ├── route.js
│       ├── config.js
│       └── shared-modules.js
└── ui
    └── assets
        ├── boostrap/
        ├── fontawesome/
        ├── jquery/
        ├── params/
```

6 - POINTS PARTICULIERS

6.1 - Gestion des logs

Les logs de l'application sont historiés quotidiennement à la date du jour et conservés sur 30 jours glissants sur le serveur pour chaque micro-service.

6.2 - Fichiers de configuration

6.2.1 - Application OC-Pizza

Les micro-services possèdent un fichier de configuration externalisé et accessible via cloud-config au nom du micro-service *.properties. Ils possèdent également un fichier de configuration bootstrap.properties permettant de définir les paramétrages.

6.2.1.1 - Fichier ms-.properties*

Les sources de donnée sont paramétrées dans le fichier .properties de cloud-config. Ce fichier contient l'ensemble des paramètres du micro-service 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 micro-service tel que le nom et le port pour l'enregistrement auprès d'Eureka Server.

6.3 - Ressources

Maven Repository : Pour la recherche et la mise à jour des bibliothèques tierces du projet.
https://mvnrepository.com/

Spring initializr : Pour l'aide à la génération de nouveaux modules ou à l'ajout de dépendences Spring.
https://start.spring.io/

Angular Material : Pour la conception material design de l'interface utilisateur.
https://material.angular.io/components/categories

Bootstrap : Pour le prototypage de l'interface utilisateur au sein de l'application Angular.
https://getbootstrap.com/docs/4.5/getting-started/introduction/

6.4 - Environnement de développement

Les développements sont réalisés avec des IDE au libre choix des développeurs.

Nous préconisons cependant l'utilisation des licences server Jetbrains mises à disposition pour **Intellij** pour le développement back-office (Java EE / SpringBoot) et **WebStorm** pour le front-office (Angular / Bootstrap).

Ou sans licences **STS4** (Spring Tools Suite 4) ou Eclipse avec les modules Spring installés + **Visual Studio Code** de Microsoft.

Les Webservice pourront être testé directement via l'interface **Swagger UI** ou des logiciels spécialisés pour les tests API tel que **Postman** ou Soap UI.

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

Les livraisons se feront sur GitLab par l'intermédiaire de **GitKraken**.

6.5 - Procédure de packaging / livraison

L'applicatif est packagé par Maven configuré dans spring au niveau du process de build.

Le jar exécutable est généré au lancement de la commande « mvn clean package » après vérification des tests unitaires.

```
<build>
	<plugins>
		<plugin>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-maven-plugin</artifactId>
		</plugin>
	</plugins>
</build>
```

Il est impératif d'effectuer un « mvn clean install » avant tout déploiement afin d'effectuer la phase de tests d'intégration.

Les livraisons sont effectuées sur l'environnement de développement (DEV) après validation de la merge request par un autre développeur dès qu'une fonctionnalité (feature) est finalisée.

Une livraison en fin de sprint est effectuée sur l'environnement de qualification (QUALIF) contenant l'ensemble des fonctionnalités finalisées pendant le sprint, après une phase de recette interne. Il s'agit d'une recette fonctionnelle et technique effectué par l'équipe de développement.

Après présentation des fonctionnalités livrées, le client est alors en charge d'effectuer la recette fonctionnelle et de remonter les bugs dans l'outil pendant la durée du sprint suivant.

A l'issue de cette phase, toutes les erreurs remontées devrons être traité, un procès-verbal de livraison seras rédigé est validé par les deux parties.

La livraison sur l'environnement de production (PROD) sera alors effectuée.