OC Pizza

Système de gestion de pizzeria

Dossier de conception technique

Version 1

Auteur
Rémi Moustey
Développeur Java

TABLE DES MATIÈRES

1 - VERSIONS

Auteur	Date	Description	Version
Rémi Moustey	03/01/2020	Création du document	1
Rémi Moustey	13/01/2020	Corrections et ajouts d'informations	2

2 - INTRODUCTION

2.1 -Objet du document

Le présent document constitue le dossier de conception technique du système de gestion de pizzeria.

Objectif du document : détailler la conception technique de l'application pour l'équipe technique de maintenance et de développement du client.

Les éléments du présents dossiers découlent :

- des diagrammes UML de composants de l'application web
- du diagramme UML de déploiement de l'application web

2.2 -Références

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

1. **DCT - Projet 8 - Dossier de conception fonctionnelle** : dossier de conception fonctionnelle de l'application.
2. **DCT - Projet 8 - Dossier d_exploitation** : dossier d'exploitation à l'attention de l'équipe technique du client.
3. **DCT - Projet 8 - PV Livraison** : le procès-verbal de livraison finale.

3 - Architecture Technique

3.1 -Composants généraux

Tous les packages sont dotés d'un composants resources qui permet de décrire le contexte du composant en question.

3.1.1 - Package Webapp

3.1.1.1 - Composant Servlets

Regroupe les différentes Servlet permettant de rediriger l'utilisateur vers la vue correspondant à l'URL demandée.

3.1.1.2 - Composant Webapp

Contient la partie front-end de l'application : le fichiers .jsp, les fichiers .css, les images et le fichier web.xml, qui permet de diriger l'utilisateur vers la servlet en fonction de l'URL entrée dans le navigateur.

3.1.2 - Package Consumer

3.1.2.1 - Composant Contract

Contient les interfaces de DAO qui devront être implémentées ainsi que la DAOFactory.

3.1.2.2 - Composant Impl

Implémente les DAO et contient une classe récupérant la dataSource (les informations de connexion à la base de données entrées dans le fichier XML de contexte).

3.1.2.3 - Composant Rowmapper

Contient les classes qui implémentent la classe RowMapper pour créer des mappers utilisés afin de travailler avec les données récupérées de la base de données.

3.1.3 - Package Model

3.1.3.1 - Composant Beans

Regroupe tous les beans (classes avec des getters et des setters) représentant les objets de l'application.

3.1.4 - Package Business

3.1.4.1 - Composant Contract

Regroupe les interfaces des managers.

3.1.4.2 - Composant Impl

Regroupe les implémentations des interfaces.

3.1.5 - Package Technical

Contient uniquement le fichier de configuration de log4j2.

3.1.6 - Package Batch

Contient le batch de l'application, dont le fonctionnement est détaillé dans le dossier de conception fonctionnelle, implémenté dans la fonction main.

3.2 -Application Web

La pile logicielle est la suivante :

- Application **J2EE** (JDK version 1.8).
- Serveur d'application **Tomcat 7**.
- Système de Gestion de Base de Données : PostgreSQL.
- Intégration en base de données via ORM avec Spring Data.
- Utilisation du moteur de templates Thymeleaf.

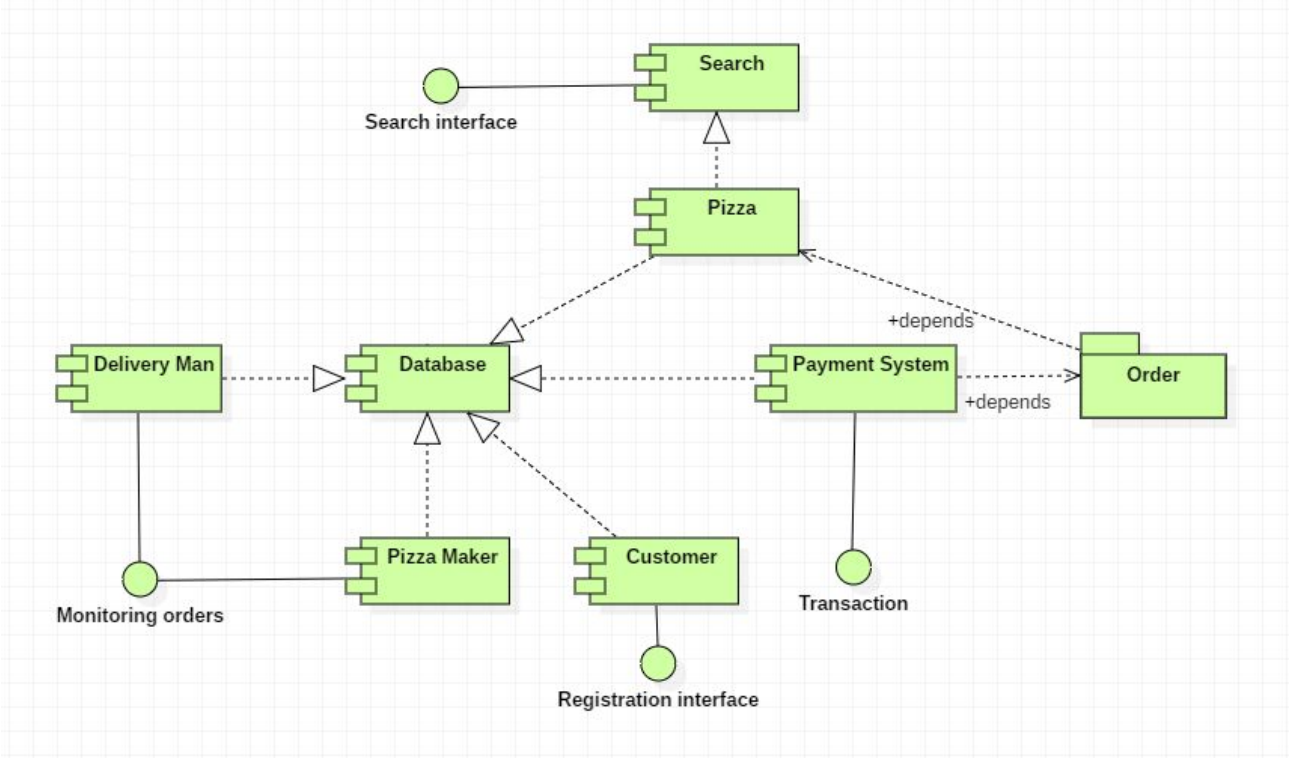

Diagramme UML de Composants

3.2.1 - Composant Database

Il s'agit de la base de données PostgreSQL qui sera utile pour regrouper toutes les données utilisées dans l'application. Les objets y seront regroupées dans les tables qui s'organiseront comme sur le modèle physique de données présent dans le fichier annexe.

3.2.2 - Composant Pizza

Chaque pizza proposée à la carte d'OC Pizza peut être présente dans une table de la base de données. De cette façon, il sera possible pour chaque client de la commander. La gestion de la liste des pizzas disponibles est à la charge des administrateurs de l'application.

3.2.3 - Composants Order et Payment System

Les pizzas sélectionnées par le client constituent une commande réalisée par le client, chaque commande pouvant être constituée d'une ou de plusieurs pizzas. Le système de paiement permet au client de payer sa commande.

3.2.4 - Composant Transaction

Il s'agit de l'interface où le client peut voir sa commande et la payer. Elle est constituée d'un formulaire où le client peut entrer ses informations de paiement.

3.2.5 - Composants Search et Search interface

Ce composant permet de rechercher des pizzas selon différents critères (nom, prix, type, ingrédients) grâce à une requête de type post. Les pizzas correspondantes sont sélectionnées dans la base de données, puis affichées dans l'interface correspondante.

3.2.6 - Composants Customer et Registration interface

Chaque utilisateur a la possibilité de s'inscrire sur l'application web, en remplissant un formulaire par ses coordonnées dans l'interface d'enregistrement. Une fois que l'utilisateur est inscrit, ses informations sont ajoutées dans une table de la base de données qui regroupe tous les différents clients s'étant inscrits de cette manière.

3.2.7 - Composants Delivery Man et Pizza Maker

Comme les clients, les employés du groupe de pizzeria sont présents dans la base de données, dans deux tables différentes selon leur rôle. Ils s'inscrivent dans un espace salarié et les administrateurs de l'application peuvent les accepter ou non. Ce statut particulier leur donne la possibilité d'interagir avec plus d'éléments qu'un client classique, sur le statut d'une commande par exemple.

3.2.8 - Composant Monitoring orders

Les salariés ont accès à cet espace particulier qui leur permet de modifier le statut d'une commande effectuée par un client, lorsqu'ils réalisent les tâches de livraison ou de préparation. Le client peut également suivre ce statut.

3.3 -Application Batch

L'application batch est déployée sur un serveur Tomcat et utilise la même base de données et les mêmes DAO que l'application web.

4 - Architecture de Déploiement

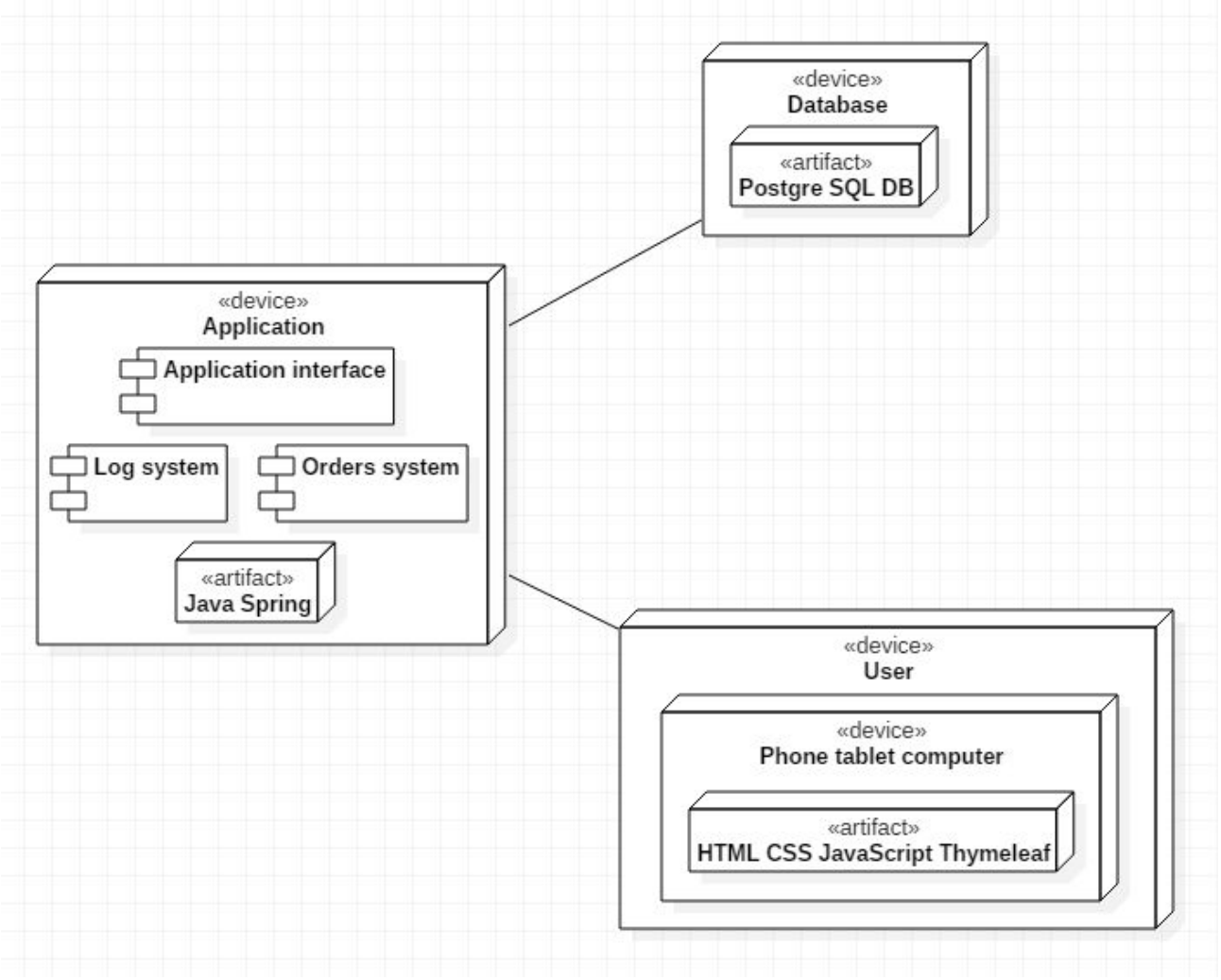

Diagramme UML de déploiement

L'ordinateur d'un utilisateur de l'application envoie des requêtes HTTP au serveur où est contenu le code Java. Ce serveur effectue, si nécessaire, des requêtes à la base de données PostgreSQL et récupère les informations dont il a besoin pour construire la page HTML que l'utilisateur a demandée. Puis, il envoie ce code HTML au navigateur de l'utilisateur qui se charge de le traduire, pour afficher la page web.

4.1 -Serveur de Base de données

Comme dit plus haut, le SGBD où est hébergée la base de données est PostgreSQL. L'interface

pgAdmin peut aussi être utilisée.

Caractéristiques techniques : Serveur Apache Tomcat 7 + PostgreSQL 12.1.

Les scripts SQL peuvent être générés à partir du modèle physique de données réalisé en amont, par l'intermédiaire de SQL Power Architect.

4.2 -Serveur d'hébergement de l'application

À voir lors du déploiement de l'application sur le Web. La version 7 du serveur Apache Tomcat sera utilisé au cours du développement pour tester le bon fonctionnement de l'application.

5 - ARCHITECTURE LOGICIELLE

5.1 -Principes généraux

Les sources et versions du projet sont gérées par **Git**, les dépendances et le packaging par **Apache Maven.**

5.1.1 - Les couches

L'architecture applicative est la suivante :

- une couche **business** : responsable de la logique métier du composant.
- une couche **model** : implémentation du modèle des objets métiers.
- une couche **consumer** : implémentation de tous les éléments d'interaction avec la base de données. Cette couche contient les DAO.
- une couche **webapp** : cette couche contient l'ensemble de l'application web et les éléments qui permettent de mettre en place l'interface front-end.
- une couche **technical** : implémentation de tous les éléments techniques comme les logs de l'application.

5.1.2 - Les modules

Les modules correspondent aux différentes couches évoquées dans le paragraphe précédent.

5.1.3 - 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 »)

```
racine
  ├─ pom.xml
  ├─ ocpizza-business
```

```
│   ├─ pom.xml
│   └─ src
│       ├─ main
│       │   ├─ java
│       │   └─ resources
│       └─ test
│           ├─ java
│           └─ resources
├─ ocpizza-model
│   ├─ pom.xml
│   └─ src
│       ├─ main
│       │   ├─ java
│       │   └─ resources
│       └─ test
│           ├─ java
│           └─ ressources
└─ src
    └─ lib
```

- Chaque répertoire source contiendra donc d'une part le code de l'application et d'autre part, les tests. La couche webapp contient en plus un dossier webapp au même niveau avec la structure suivante :

```
webapp
 ├─ css
 ├─ img
 ├─ jsp
 ├─ WEB-INF
     ├─ web.xml
```

5.2 -Application Web

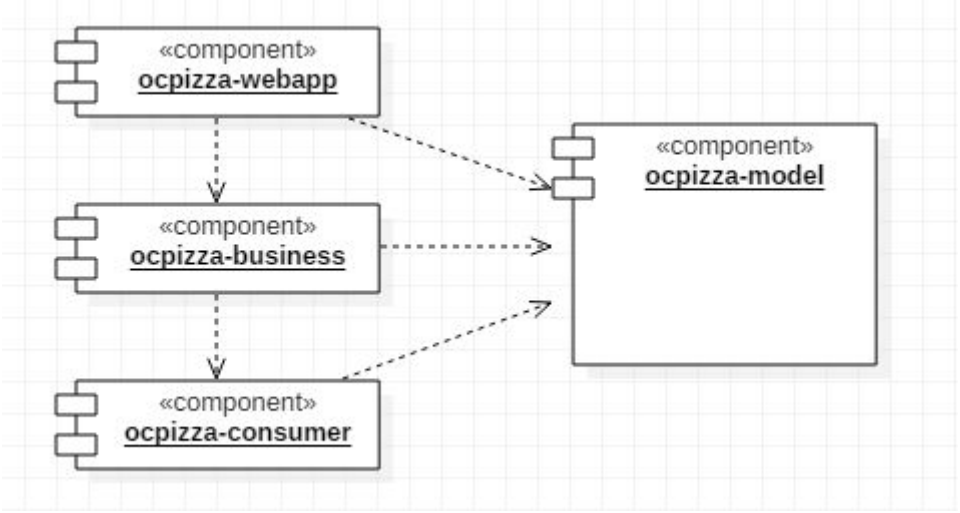

diagramme UML de composants

Le diagramme ci-dessus montre les interactions entre les différents modules, ce qui permet de noter que l'application web fait appel à la couche business qui dispose de la logique métier et qui fait elle-même appel à la couche consumer gérant les interactions avec la base de données grâce aux beans, utiles dans l'ensemble des couches.

6 - POINTS PARTICULIERS

6.1 -Gestion des logs

Durant le développement de l'application, un logger issu de log4j peut être utilisé. Les logs seront situés à la racine du projet principal dans un dossier logs. Il suffira de créer le logger en tant qu'attribut d'une classe à l'aide de la classe LogManager.

6.2 -Fichiers de configuration

6.2.1 - Application web

Différents fichiers de configuration doivent être mis en place avant l'implémentation de l'application web.

6.2.1.1 - Datasources

Les données de connexion à la base de données doivent être regroupées dans un fichier .properties. Il sera alors possible de les récupérer grâce à la fonction PropertiesFactoryBean

de Spring dans le fichier XML de contexte du module.

6.2.1.2 - Fichier log4j2.xml

Présent dans le dossier resources du module, il est utile pour configurer l'écriture des logs de l'application.

6.2.1.3 - web.xml

Comme montré sur le schéma de la partie précédente, ce fichier se trouve dans le dossier WEB-INF de la couche webapp. Il permet de déclarer des servlets et d'associer ces servlets à une URL. En clair, il est indispensable pour permettre la navigation de l'utilisateur de l'application web entre les différentes pages.

6.2.2 - Application Batch

6.2.2.1 - Datasources

Comme pour l'application web, les données de connexion à la base de données sont regroupées dans un fichier .properties, auquel on accède simplement via la fonction PropertiesFactoryBean de Spring dans le fichier XML du contexte du module.

6.2.2.2 - Fichier log4j2.xml

Ce fichier permet de configurer les logs de l'application.

6.2.2.3 - Fichier config.properties

Ce fichier de configuration contient le chemin vers le répertoire qui regroupe les factures des commandes des clients.

6.3 -Ressources

Le dossier resources se constitue des fichiers de contexte. Ces fichiers XML peuvent être utiles pour récupérer les beans, afin de pouvoir bénéficier de l'injection de dépendances. Par exemple, dans la couche webapp, il est utile de créer des beans DAO et Manager.

6.4 -Environnement de développement

Le développement de l'application s'effectuera grâce à la version propriétaire de l'environnement de développement IntelliJ IDEA.

6.5 -Procédure de packaging / livraison

Lorsque la connexion à la base de données sera établie, il suffira de se placer à la racine du projet et d'exécuter la commande *mvn clean install tomcat7:run*. Elle permet de créer les dossiers target contenant les fichiers .class. Il est à noter que le *mvn clean install* peut être

réalisé grâce à IntelliJ IDEA. Il suffit de cliquer sur l'onglet "Maven", en haut à droite de la fenêtre, d'ouvrir le dossier "Lifecycle", puis de sélectionner les options correspondantes et de cliquer sur "Run Maven Build" dans le menu supérieur.