OC Pizza

Système de gestion de pizzeria

Dossier d'exploitation

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 d'exploitation du système de gestion de pizzeria.

Objectif du document : informer l'équipe d'exploitation sur la manière d'exploiter le système et de réagir face à d'éventuels problèmes rencontrés.

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 de conception technique** : dossier de conception technique de l'application.
3. **DCT - Projet 8 - PV Livraison** : le procès-verbal de livraison finale.

3 - PRÉ-REQUIS

3.1 -Système

3.1.1 - Serveur de Base de données

Le serveur de base de données héberge la base de données *oc_pizza* qui comporte les schémas *point_of_sale*, *creation_order* et *management_order*.

3.1.1.1 - Caractéristiques techniques

Le SGBD choisi étant PostgreSQL, cette base de données est largement multiplateforme. Tous les types de données nécessaires pourront être stockés dans cette base de données.

3.1.2 - Serveur Web

Serveur physique ou virtuel hébergeant l'application web.

3.1.2.1 - Caractéristiques techniques

Pour le développement, la version 7 du serveur Apache Tomcat est utilisée. Il sera exécuté vie la commande *mvn tomcat7:run*. Le port par défaut de ce serveur est *8080*.

3.1.3 - Serveur de Batches

Le serveur de batches sera également un serveur Apache Tomcat. Il sera exécuté tous les jours, aux heures d'ouverture des points de vente et permettra de notifier le client sur le statut de sa commande.

3.1.4 - Serveur de Fichiers

Ce serveur sera utilisé grâce au protocole FTP. C'est sur ce serveur que seront stockées les factures des clients, qui seront générées au fur et à mesure des commandes. Bien sûr, chaque client ne doit avoir accès qu'à ses propres factures.

3.2 -Bases de données

Les bases de données et schémas suivants doivent être accessibles et à jour :

- **base *oc_pizza* :** version 1
- **schéma *point_of_sale*** : version 1
- **schéma *creation_order*** : version 1
- **schéma *management_order*** : version 1

3.3 -Web-services

Les web services suivants doivent être accessibles et à jour :

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

4 - Procédure de déploiement

4.1 -Déploiement des Batches

4.1.1 - Artefacts

Les batches du système de gestion de pizzeria sont construits sous la forme d'une archive ZIP contenant les répertoires :

- **bin** : les scripts SH de lancement des différents batches
- **conf** : les fichiers de configuration
- **main** : les fichiers java d'exécution des batches, selon la structure habituelle (main/java/package).
- **resources** : les fichiers de configuration

Extraire l'archive **XXX.zip** dans le répertoire :

/oc_pizza/batch

Positionner les droits d'exécution sur les scripts SH de lancement des batches.

Intégrer un logger de log4j2 à l'aide d'un fichier de configuration comme expliqué dans la section 4.1.3.

4.1.2 - Variables d'environnement

Voici les variables d'environnement reconnues par les batches du système de gestion de pizzeria :

Nom	Obligatoire	Description
PATH_HOME	non	Répertoire racine de l'installation de l'application
PATH_BILLS	oui	Répertoire où seront regroupées les factures

Définir les variables d'environnement dans un fichier config.properties qui se situe dans le répertoire conf du batch.

4.1.3 - Configuration

Voici les différents fichiers de configuration :

- **config.properties** : fichier de constantes
- **spring_configuration.properties (non obligatoire)** : fichier éventuel de constantes pour la connexion à la base de données

4.1.3.1 - Fichier config.properties

Il regroupe les constantes tirées des variables d'environnement.

4.1.3.2 - Fichier spring_configuration.properties

Ce fichier éventuel permettrait de regrouper toutes les informations de connexion à la base de données et qui peut être lu dans le fichier batchContext.xml.

4.1.4 - Ressources

Voici les différents fichiers du dossier *resources* des batches :

- **log4j2.xml** : fichier de configuration des logs
- **batchContext.xml** : fichier de configuration du système
- **bootstrapContext.xml** : autre fichier de configuration du système

4.1.4.1 - Fichier log4j2.xml

Ce fichier est à intégrer dans le dossier resources, afin de définir la façon dont les logs seront écrits.

4.1.4.2 - Fichier batchContext.xml

Il permet de définir les différents beans, qui sont les DAO, la dataSource avec les données de connexion à la base de données.

4.1.4.3 - Fichier bootstrapContext.xml

Il permet d'importer les autres fichiers XML de contexte dans le module batch.

4.1.5 - Vérifications

Afin de vérifier le bon déploiement des batches, exécuter un test, vérifiant que l'envoi d'une notification vers un client imaginaire fonctionne correctement.

4.2 -Déploiement de l'Application Web

4.2.1 - Artefacts

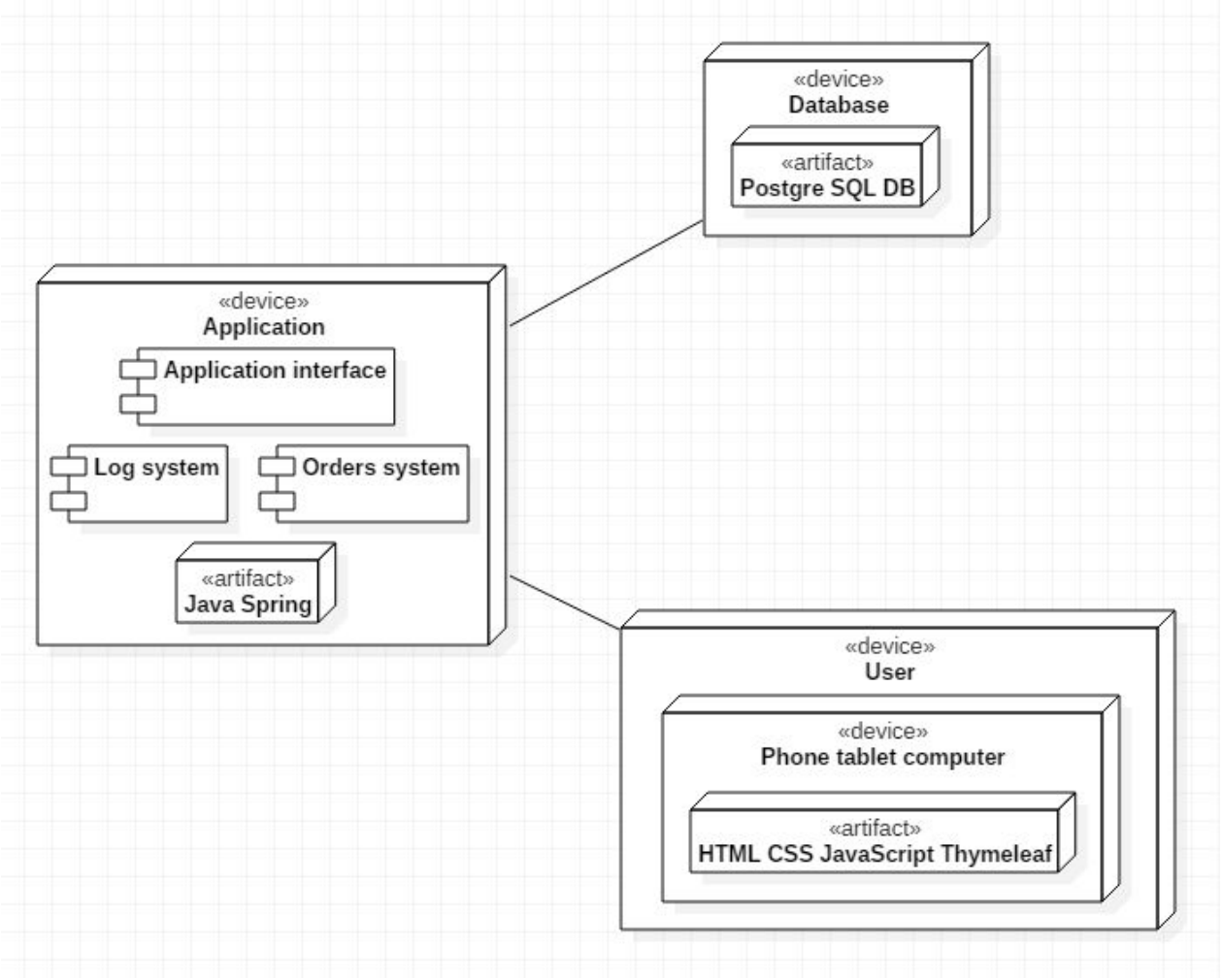

Diagramme de déploiement de l'application

Script de création de la base de données et du jeu de données :
https://github.com/RemiMoustey/SQL_OCPizza

4.2.2 - Environnement de l'application web

4.2.2.1 - Variables d'environnement

Le serveur d'application Tomcat doit être exécuté avec la variable d'environnement suivante définie au démarrage. Elle est nécessaire afin de récupérer le répertoire contenant les fichiers de configuration de l'application :

```
-Dcom.ocpizza.apps.conf=$home_application_conf_directory
```

INFO : il ne faut pas mettre de « / » à la fin de la valeur de la variable et ne pas utiliser d'espace dans le chemin.

4.2.3 - Répertoire de configuration applicatif

Le répertoire de configuration applicatif doit être créé sur le système de fichier et définit de la façon suivante :

```
$home_application_conf_directory/ocpizza
```

Les fichiers de configuration sont les suivants :

- web.xml
- context.xml

4.2.3.1 - Fichier web.xml

Il se situe dans le dossier WEB-INF et permet de déclarer une servlet, puis de la lier à une URL. Les vues liées à l'URL correspondante sont également déclarées dans ce fichier.

4.2.3.2 - Fichier context.xml

Il se situe dans le dossier META-INF. Il est utilisé par Tomcat pour configurer le déploiement de l'application via Tomcat.

4.2.4 - DataSources

Les accès à la base de données doivent se configurer à l'aide du fichier webappContext.xml, ou d'un fichier properties qui sera lu dans ce fichier webappContext.xml.

Le fichier de drivers **postgresql (postgresql-12.1)** doit être déposé dans le répertoire :

```
$home_server/lib/ext
```

Bien sûr, les dépendances doivent être incluses dans le pom.xml.

4.2.5 - Ressources

Voici les autres fichiers du dossier *resources* de l'application web :

- **webappContext.xml** : fichier de configuration du système
- **bootstrapContext.xml** : autre fichier de configuration du système

4.2.5.1 - Fichier webappContext.xml

Comme évoqué dans la partie précédente, il est utile pour regrouper les accès à la base de données, mais aussi pour déclarer tous les beans nécessaires au fonctionnement de l'application comme les DAO, les Managers.

4.2.5.2 - Fichier bootstrapContext.xml

Il permet d'importer les autres fichiers XML de contexte dans le module webapp.

4.2.6 - Vérifications

Afin de vérifier le bon déploiement de l'application, les tests sont nécessaires. Le développement peut d'ailleurs s'effectuer en TDD. Des tests doivent être écrits pour chacune des fonctionnalités, tout en s'assurant qu'ils couvrent bien tous les fonctionnements possibles.

5 - Procédure de démarrage / arrêt

5.1 -Base de données

Démarrage : Il est possible de démarrer le serveur de connexion à la base de données, par l'intermédiaire du programme postgres, qui utilisera certaines données. Il faut donc également préciser les données qu'il doit utiliser avec l'option -d, de la manière suivante :

```
$ postgres -d /usr/local/pgsql/data
```

Plus de détails sont disponibles ici : https://docs.postgresql.fr/8.4/server-start.html

Arrêt : De la même manière, un arrêt rapide peut être envoyé par cette commande :

```
$ kill -int `head -1 /usr/local/pgsql/data/postmaster.pid`
```

Plus de détails sont disponibles ici : https://docs.postgresql.fr/8.4/server-shutdown.html

5.2 -Batches

Démarrage : il suffit d'exécuter la fonction main de la classe qui la contient, à l'aide d'IntelliJ

IDEA ou en ligne de commandes.

Arrêt : il suffit de stopper cette fonction main.

5.3 -Application web

Démarrage : il est possible de démarrer l'application web simplement en ligne de commande, en se plaçant dans le dossier racine du projet et en démarrant le serveur Tomcat 7 importé par une dépendance Maven de cette manière : `mvn tomcat7:run`.

Arrêt : il suffira de stopper le serveur Tomcat 7 lancée par la commande de démarrage.

6 - Procédure de mise à jour

6.1 -Base de données

Si la base de données doit être altérée, elle peut l'être lors de la phase de développement afin d'ajouter ou de modifier des champs, d'ajouter des clés étrangères, en ligne de commandes ou via l'interface pgAdmin.

6.2 -Batches

La fonction main ou les classes utilisées dans cette fonction peuvent être modifiées à tout moment, mais pour que les modifications soient prises en compte, il sera nécessaire de redémarrer la fonction main comme expliqué dans la partie précédente.

6.3 -Application web

Toutes les modifications apportées à la couche webapp ou aux couches utiles au bon fonctionnement de l'application web (exception faite des vues) nécessite une recompilation des sources. Il sera donc nécessaire d'arrêter le serveur Tomcat, puis d'exécuter la commande suivante pour effectuer un redémarrage avec la prise en compte de toutes les mises à jour :

```
mvn clean install tomcat7:run
```

7 - Supervision/Monitoring

7.1 -Supervision de l'application web

Afin de tester que l'application web est toujours fonctionnelle, il est recommandé de la superviser à l'aide de Zabbix, en exécutant périodiquement un scénario de navigation métier. Il est également utile de créer un scénario fonctionnel avec Jmeter, comme expliqué dans ce billet de blog : https://blog.syloe.com/superviser-une-application-web-zabbix-jmeter/

8 - Procédure de sauvegarde et restauration

Des commits quotidiens devront être faits sur le dépôt local GitHub. Évidemment, des branches devront être créées lors des différents sprints, en prenant garde qu'il n'y ait pas de risque de conflits lors de la fusion avec la branche master. Ces commits réguliers permettront de maintenir le dépôt à jour, afin de disposer d'une sauvegarde complète de fichiers de développement.

Il est aussi nécessaire de préparer la sauvegarde de la base de données PostgreSQL. Les scripts seront disponibles sur un dépôt GitHub autre que celui du code source, en pensant à les mettre à jour, à l'aide d'un dump quotidien. La procédure de réalisation d'un dump avec PostgreSQL est détaillée ici : https://www.postgresql.org/docs/9.1/backup-dump.html

Pour restaurer, il suffira ensuite de :

1. Effectuer un clone du dépôt GitHub.
2. Créer une base de données PostgreSQL et importer les scripts SQL clonés.
3. Réaliser les procédures de démarrage détaillées en partie 5.