OC PIZZA

Pizzeria OC

Dossier de conception technique

Version 1.0

Auteur
Enyo H. Tovissou
Analyste Développeur

TABLE DES MATIÈRES

1 - VERSIONS

Auteur	Date	Description	Version
E.T	22/11/2019	Création du document	01

2 - INTRODUCTION

2.1 - Objet du document

Le présent document constitue le dossier de conception technique de l'application Pizzeria OC.

Objectif du document est de présenter les outils, les technologies, et les méthodes mises en œuvre pour réaliser l'application Pizzeria OC.

Les éléments du présent dossier découlent :

- de l'entretien réalisé avec le responsable du groupe OC PIZZA du 22.11.2019
- de analyse des besoins et de la rédaction du dossier de conception fonctionnelle par le l'équipe de It-Consutling & Development.

2.2 - Références

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

1. DCF – 1.0 : Dossier de conception fonctionnelle de l'application PIZZERIA OC
2. DE - 1.0 : Dossier exploitation de application PIZZERIA OC

3 - ARCHITECTURE TECHNIQUE

3.1 - Les composants généraux

Pour réaliser l'application Pizzeriaoc, nous avons réalisé un découpage en composant qui est le suivant :

3.1.1 - Composants Authentification

L'application Pizzeriaoc est une application client/serveur. Elle permet à tout utilisateur de s'authentifier avant certains accès. Par exemple un client de la Pizzéria peut voir les offres du système mais pour commander un Pizza, il aura besoin d'une authentification. De même, le patron pour consulter le stock des ingrédients, mettre à jour la liste de l'offre ou réaliser les aides mémoire a besoin d'une authentification.

3.1.2 - Composants Staff et Customer

Le composant « Staff » regroupe toutes les fonctionnalités des membres du staff notre application que sont gestion des commandes, gestion du stock d'ingrédients, réalisation des aide-mémoires. Ce composant utilise les services du composant authentification pour pouvoir fonctionner.

Le composant « Customer » regroupe les fonctionnalités liées à l'acteur client de notre système. Il permet au client donc de passer une commande, de suivre en temps réels l'évolution de sa commande et de pouvoir annuler sa commande dans la mesure du possible.

Ces composants sont réalisés eux même à partir des composants de l'architecture JSF respectant les recommandations J2EE, le MVC (Modèle, Vue et Contrôleur).

3.1.3 - Composant account

Le composant Account est strictement lié au composant Customer. Elle représente non seulement un compte utilisateur mais aussi les moyens de payement d'un utilisateur donné. Il gère certaines informations liées au compte utilisateur comme ses commandes,

3.1.4 - Composant management.

Le composant management un composant qui permet de gérer les commandes, de sa réception jusqu'à sa livraison au client, en passant sa préparation les Pizzaiolo. Il lié au composant staff et stock produit. Il est un des composants centraux pour notre application.

3.1.5 - Composant stock produit.

La gestion des commandes se fait essentiellement à base des ingrédients en stock. Quand des ingrédients manquent au stock la liste des Pizza proposées est modifiée par patron de l'entreprise. Ainsi, pour gérer au mieux les stocks et faciliter la mise à jour de la carte de la Pizzeria, nous avons compostant stock produit, qui

permet de mettre à disposition des Pizzaiolo les ingrédients souhaités, et d'actualiser les quantités du stock. Il permet aussi au patron, de suivre l'évolution du stock afin de prendre des mesures qui s'imposent.

3.1.6 - Composant order

C'est le composant qui s'occupe des commandes selon sa quantité, gérer le statu des commandes, si annulé ou pas, si déjà préparé ou pas ou encore si déjà livrée ou pas.

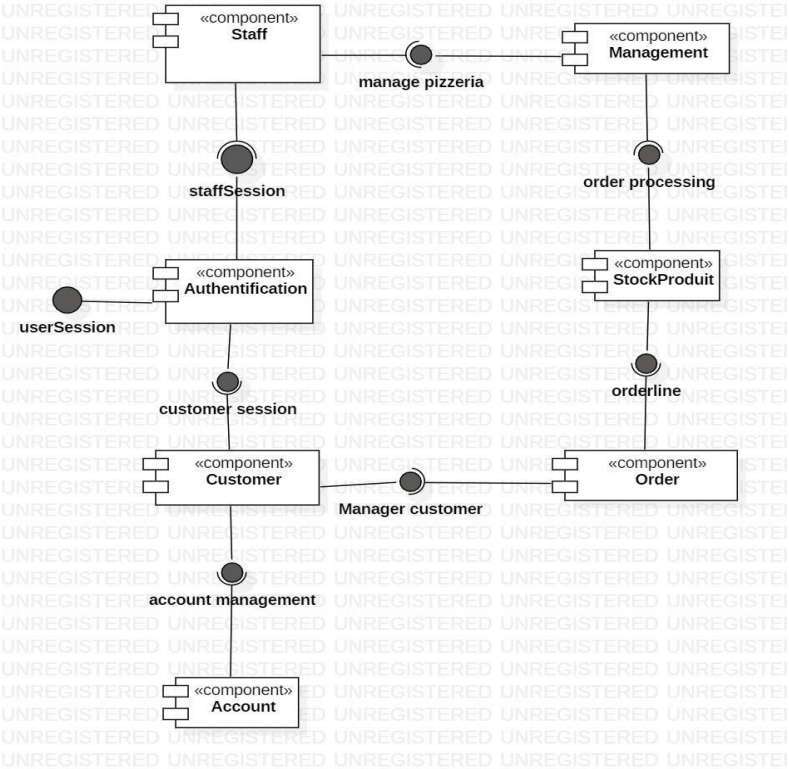


Diagramme UML de Composants

3.2 - L'application web

Le Java Server Faces JSF est une recommandation et donc a plusieurs implémentations qui tous respectent les recommandations J2EE. Dans notre application, nous utiliseront la version de Majora, qui est le JSF2.2.

Les applications JSF sont basées sur la structure MVC (Modèle, Vue et Contrôleur).

3.2.1 - Model

Managed Bean ou Backing Bean.

Le modèle représente les classes java spécialisées qui synchronisent les valeurs avec les composants de l'interface utilisateur (UI), et accède à la logique métier et gère la navigation entre les pages.

3.2.2 - La vue

Pages web (JSP, XHTML)

Ce sont les pages web de l'application. JSF utilisent les facelets, qui sont formées d'une arborescence de composants UI.

3.2.3 - Le contrôleur

Faces Servlet

Nous avons un Servlet principale de l'application qui sert de contrôleur.

Toutes les requêtes de l'utilisateur passent systématiquement par elle, qui les examines et appelle les différentes actions correspondantes.

3.2.4 - Faces-config

Il y a ensuite un fichier de configuration de l'application, le Faces-config, définissant les règles de navigation et les différentes Managed Beans utilisés

3.2.5 - Le moteur du rendu

Il se charge de décoder la requête de l'utilisateur pour initialiser les valeurs du composant et encode la réponse pour créer une représentation du composant pour le client.

3.2.6 - Convertisseurs et Validateurs

Permet de valider les champs de saisie textuelle et de les convertir vers d'autre type.

.

4 - ARCHITECTURE DE DÉPLOIEMENT

L'application Pizzeria OC a une architecture de déploiement qui se présente comme suit :

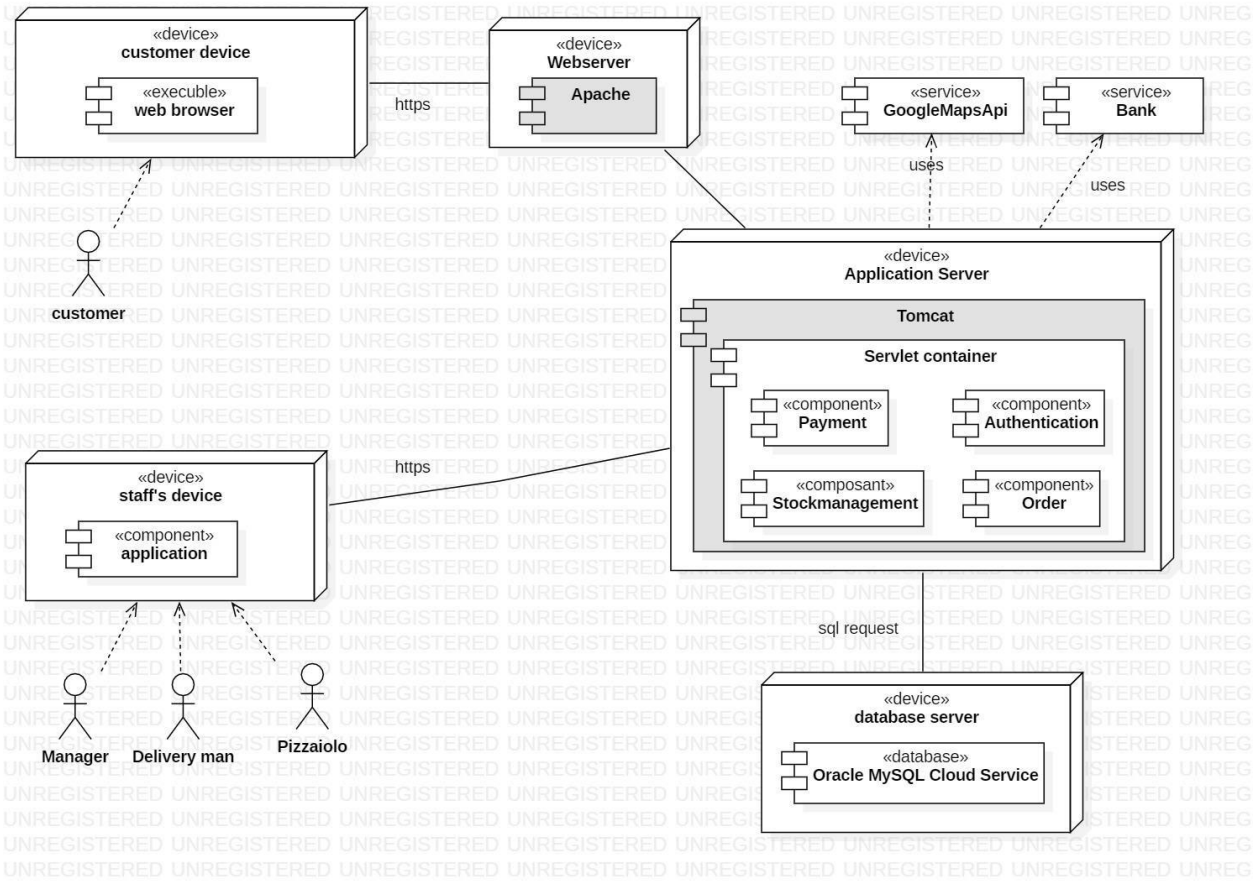


Diagramme UML de déploiement

Nous avons montré à travers ce diagramme, le déploiement des composants de notre système de gestion du groupe de Pizzeria. L'implémentation de notre application se fera donc avec le framework JSF, et le déploiement se fera sur Apache Tomcat.

Le choix de s'est porté sur JSF, car elle offre des modules bien adaptés à la réalisation des services web. Et pour le serveur, nous avons préféré un serveur virtuel Tomcat. La raison est que pour un jeune groupe, certes en plein essors, il faut minimiser un les coûts, et Tomcat avec les services qu'elle offre nous semble raisonnable. Aussi, Tomcat est largement utilisé ce qui nous fait attester sa fiabilité.

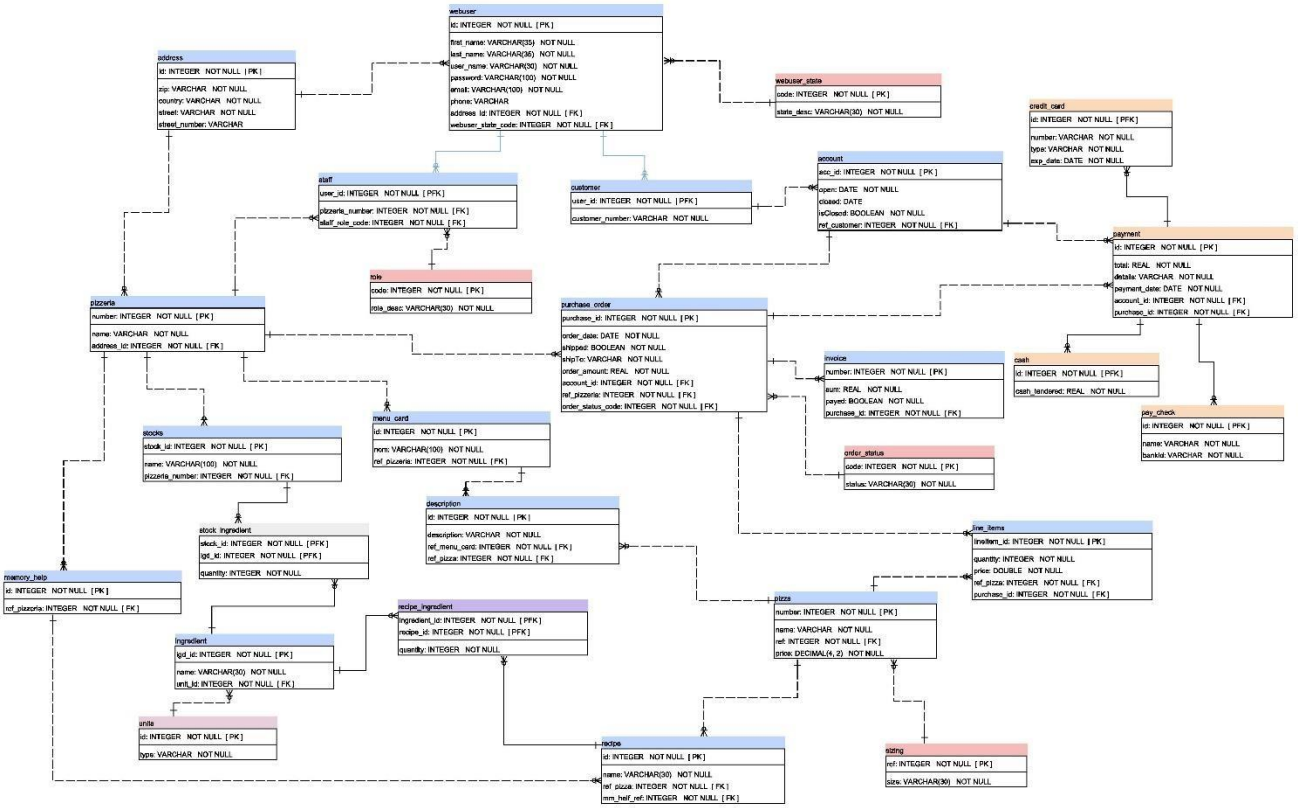
Pour la base donnée, MySQL étant largement utilisé dans le domaine web, nous somme séduit par son service cloud qui offre des services assez remarquable entre autre dans le domaine de la sécurité, la disponibilité des services et une très bonne assistance technique.

4.1 - Serveur de Base de données

Le Système de gestion de base de données (SGBD) utilisé sera PostgreSQL. Ce choix se justifie non seulement en raison de sa compatibilité avec la plateforme de déploiement, mais aussi par flexibilité et surtout parce qu'il est open Source.

4.1.1 - Le modèle physique de données

Le modèle physique de données ci-dessous est celle de l'application Pizzeria OC et est extrait du Dossier de conception fonctionnelle.



4.1.2 - La base de données

Nous l'avons mentionné un peu plus haut que nous utiliserons comme base de données le PostgreSQL et que notre application tournera sur un serveur tomcat d'apache.

PostgreSQL est système de gestion de base de données relationnelle objet (SGBRO) dont la flexibilité ne se reflète pas seulement dans sa fonctionnalité et son extensibilité, mais offre également de nombreuses possibilités de configuration logicielle et matérielle. Postgres est notamment inclus par défaut dans la plupart des distributions UNIX/Linus.

Grâce à des packs d'installation appropriés, les systèmes d'exploitation Windows peuvent également être sélectionnés comme plateformes pour le système. La puissance de calcul et la capacité de stockage requises dépendent uniquement de la taille du système de base de données prévu, le logiciel libre lui-même nécessite un peu moins de 20 Mo.

Postgres est basé sur le modèle client-serveur classique : le composant serveur central « postmaster » gère tous les fichiers de base de données ainsi que toutes les connexions établies pour la communication avec le serveur de base de données. Les utilisateurs n'ont besoin que d'un programme client approprié pour établir la connexion, alors que le progiciel PostgreSQL avec psql a déjà intégré une solution native pour fonctionner via la ligne de commande ou le terminal. Alternativement, on peut passer à diverses applications avec des interfaces utilisateurs graphiques, par exemple le pgAdmin.

Dans notre cas, nous voulons installer PostgreSQL sur Apache-Tomcat, qui une fois configuré, permettra à l'utilisateur de faire une requête web au Serveur Web d'Apache, qui le reconnaitra comme ne demande de servlet à gérer par Tomcat. Ce dernier, qui est un conteneur de servlet Java et de moteur JSP, exécutera le servlet Java qui utilisera JDBC pour accéder à la base de données PostgreSQL.

Le servlet générera dynamiquement une page Web en fonction des résultats de la requête de base de données et transmettra ces résultats à Apache httpd, qui renverra le contenu Web au navigateur client.

4.1.3 - PostgreSQL sur Apache Tomcat : configuration

Nous assumons que Apache Tomcat est déjà installé et configuré sur la machine avec `$CATALINA_HOME` qui indique le chemin d'accès de l'installation de Tomcat.

Il faut donc télécharger le fichier ***postgresql.jar*** et le mettre dans `$CATALINA_HOME/commun/lib` et ensuite faire quelques modification dans le

`<Context>` ….`</Context>` du fichier `conf/server.xml`.

La page ***https://jdbc.postgresql.org/documentation/head/tomcat.html***l explique plus en détails sur la configuration de postgresql dans Tomcat.

4.2 - Serveur de déploiement

Comme mentionné plus haut, nous utiliserons Tomcat, conteneur d'Apache qui est un conteneur web libre de servlets et JSP JAVA EE pour le déploiement notre application. Il implémente les spécifications des servlets et des JSP du JAVA Community Process, et est paramétrable par des fichiers XML et inclut des outils pour sa configuration et sa gestion. Il comporte également un serveur htttp qui est lui-même écrit en Java. Le nombre d'utilisation de Tomcat dans le monde des serveurs domine largement celle de JBoss et Jetty. Ce qui nous confirme qu'il fait office d'un bon choix.

Pour plus d'information à ce sujet, consulter le site `http://tomcat.apache.org/`

5 - ARCHITECTURE LOGICIELLE

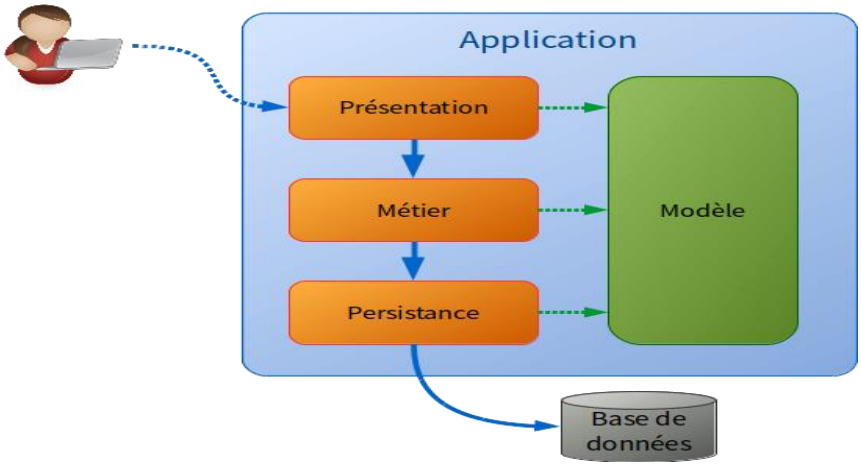
5.1 - Les principes généraux

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 persistance responsable de la persistance du model dans la base de données
- une couche présentation, qui contient la vue et les contrôleurs et sur lesquels interagissent les utilisateurs de l'application.



Architecture multi-tiers

Cette image présente les différentes couches et les flèches indiquent comment les appelles s'effectuent entre les couches. Une couche située plus haut, appelle une couche située plus bas, et la couche basse n'a aucune information de la couche plus haute.

5.1.2 - Les modules

En utilisant Maven dans notre application Pizzeria OC, nous pouvons la moduler de la manière suivante :

- Un module webapp
- Un module journalisation ou logs,
- Un module business
- Un module consommateur
- Un module Model

Le module webapp et journalisation, tous deux utilisent le model, et business. Et le business quant à lui utilise le module consommateur (consumer en anglais) pour pouvoir persister des données du Model dans la base de données. Le consumer aussi utilise le module Model.

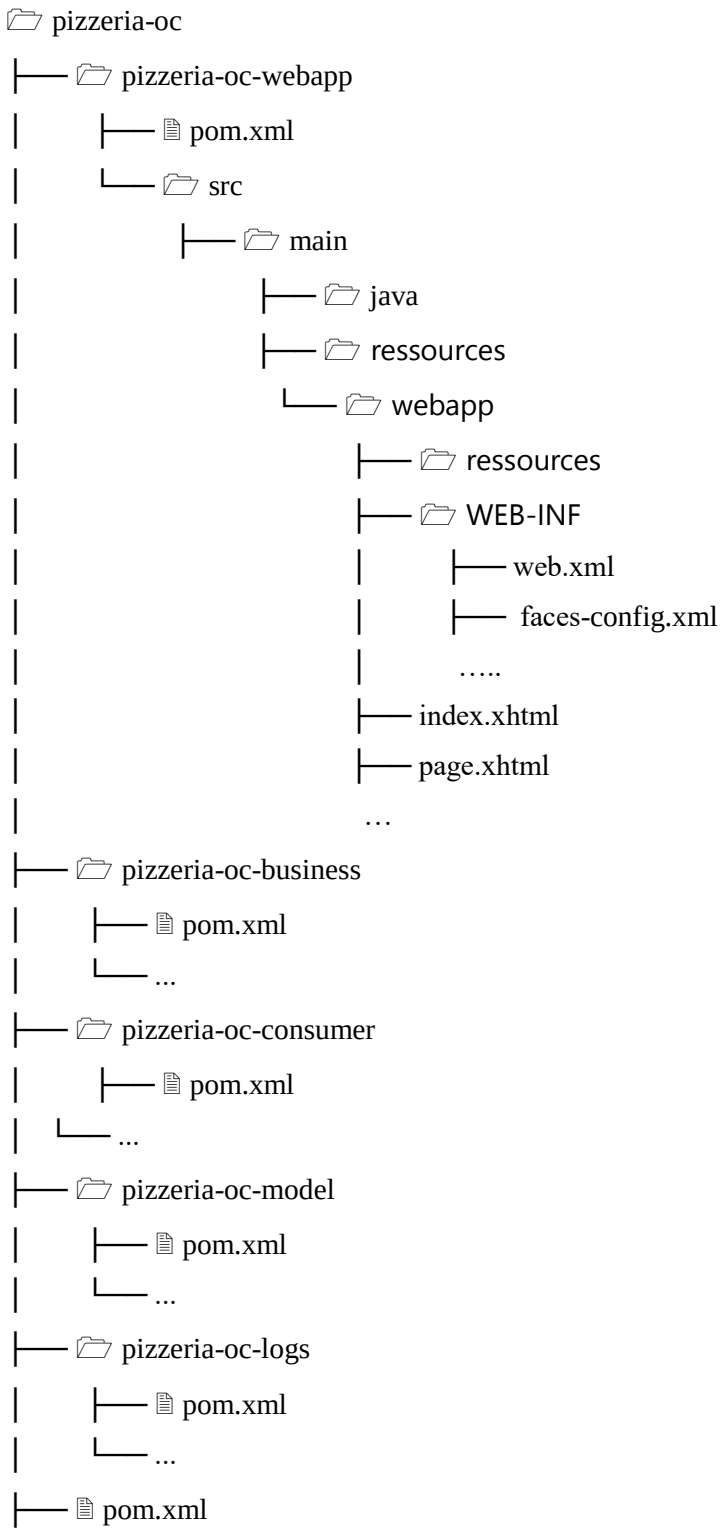
5.1.3 - Structure des sources

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 »)


OC PIZZA


It-Consulting & Development


pizzeria-oc
pizzeria-oc-webapp
pom.xml
src
main
java
ressources
webapp
ressources
WEB-INF
web.xml
faces-config.xml
…..
index.xhtml
page.xhtml
…
pizzeria-oc-business
pom.xml
...
pizzeria-oc-consumer
pom.xml
...
pizzeria-oc-model
pom.xml
...
pizzeria-oc-logs
pom.xml
...
pom.xml

6 - POINTS PARTICULIERS

6.1 - Gestion des logs

La journalisation se fait généralement dans le de Tomcat dans

`${catalina.base}/confing/logging.properties`.

Le fichier est spécifié par la propriété système `java.util.logging.config.file` qui est définie par les scripts de démarrage.

Dans le cadre d'une application Web comme la nôtre, le fichier sera :

`WEB-INF/classes/logging.properties`

Il faut noter que fichier `conf/logging.properties` par défaut dans Apache Tomcat ajoute également plusieurs **AsyncFileHandlers** qui écrivent dans les fichiers.

Le seuil de niveau de journal d'un gestionnaire est INFO par défaut et peut être défini en utilisant SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST ou ALL. On peut également cibler des paquets spécifiques pour collecter les logs et spécifier un niveau.

Pour activer la journalisation de débogage pour une partie des fonctions internes de Tomcat, vous devez configurer le(s) logger(s) approprié(s) et le(s) gestionnaire(s) approprié(s) pour utiliser le niveau FINEST ou ALL, par ex :

```
org.apache.catalina.session.level=ALL

java.util.logging.ConsoleHandler.level=ALL
```

6.2 - Fichier de configuration

6.2.1 - Application web

Application Pizzeria OC étant une application JSF, elle possède deux fichiers de configuration principales qui contenir les informations nécessaires à sa bonne exécution. Nous avons le fichier web.xml contenu dans le répertoire WEB-INF, qui est un fichier descripteur de toute application web J2EE, et le fichier de configuration au format XML, qui est particulier au paramétrage de JSF, le faces-config.xml.

6.2.2 - Data sources

L'application Pizzeria OC étant une application JSF, elle respecte les spécifications de J2EE. Et en tant que tel, elle a la structure définie par J2EE pour toutes les applications web.

Le fichier web.xml doit contenir au minimum certaines informations notamment sur la servlet qui fait office de contrôleur, ses mapping des URL, et quelques paramètres comme indiqué ci-après au point 6.2.1.2.

/
/WEB-INF
/WEB-INF/web.xml
/WEB-INF/lib
/WEB-INF/classes

Le fichier web.xml doit contenir au minimum certaines informations notamment, la servlet faisant office de contrôleur, le mapping des URL pour cette servlet et des paramètres.

Chaque implémentation de JSF nécessite un certain nombre de bibliothèque tierce pour son bon fonctionnement. Par exemple, pour l'implémentation de référence, bibliothèques tierces suivantes sont nécessaire :

- jsf-api.jar
- jsf-ri.jar
- jstl.jar
- standard.jar
- common-beanutils.jar
- commons-digester.jar
- commons-collections.jar
- commons-logging.jar

Les fichiers nécessaires dépendent de l'implémentation utilisée.

Ces bibliothèques peuvent être mises à disposition de l'application selon plusieurs modes :

- incorporées dans le package de l'application dans le répertoire /WEB-INF/lib
- incluses dans le répertoire des bibliothèques partagées par les applications web des conteneurs web s'ils proposent une telle fonctionnalité. Par exemple avec Tomcat, il est possible de copier ces bibliothèques dans le répertoire shared/lib.

L'avantage de la première solution est de faciliter la portabilité de l'application sur différents conteneur web mais elle duplique ces fichiers si plusieurs applications utilisent JSF.

Les avantages et inconvénients de la première solution sont exactement à l'opposé de ceux de la seconde solution. Le choix de l'une ou l'autre est donc à faire en fonction du contexte de déploiement.

6.2.3 - Fichier web.xml

Le fichier web.xml contenu dans le WEB-INF est un fichier descripteur de toute application web J2EE et doit de ce fait contenir au minimum certaines informations sur la servlet servant de contrôleur, sur les mapping des url, ainsi que des paramètres pour configurer JSF.

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
 <display-name> Pizzeria OC </display-name>
 <description>Application de gestion de Pizzeria </description>
 <context-param>
      <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
      <param-value>client</param-value>
 </context-param>
<!-- Servlet servant de controleur-->
<servlet>
  <servlet-name>FacesServlet</servlet-name>
   <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
<load-on-startup> 1 </load-on-startup>
</servlet>
<!-Le mapping de la servlet -->
  <servlet-mapping>
     <servlet-name>Faces Servlet</servlet-name>
     <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
</web-app>
```

- Le paramètre de contexte `javax.faces.STATE_SAVING_METHOD` permet de préciser le mode d'échange de l'état de l'arbre des composants de la page. Il y a deux valeurs possibles que sont le client et le serveur.
- Le tag <servlet> permet de définir une servlet et de préciser qu'elle sera utilisée comme contrôleur dans l'application. Le plus souvent, on utilise la servlet fournie avec l'implémentation de référence, le `javax.faces.webapp.FacesServlet`.
-
- Le tag <load-on-startup> avec la valeur 1 permet de demander le chargement de servlet au lancement de l'application.
- Le tag <servlet-mapping> permet quant à elle de préciser le mapping des URLs qui seront traitées par la servlet.

  On peut avoir pour mapping <url-pattern>*.faces<url-pattern> ou

  <url-pattern>/faces/*<url-pattern>

Les URL utilisés pour des pages mettant en oeuvre JSF doivent obligatoirement passer par cette servlet. Ces URL peuvent être de différentes formes selon le mapping défini. On peut donc avoir selon les cas :

http://localhost:8080/pizzeriaoc/index.faces ou http://localhost.8080/pizzeriaoc/faces/index.xhtml.

Dans les deux cas, c'est la servlet utilisée comme contrôleur qui va déterminer le nom de la page JSF à utiliser. Notons ici l'utilisation du suffixe .xhtml, qui doit être défini sous forme :

```
< context->
        <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
         <param-value>.xhtml</param-value>
</content-param>
```

Le démarrage d'une application directement avec une page par défaut utilisant JSF ne fonctionne pas correctement. Il est préférable d'utiliser une page HTML qui va effectuer une redirection vers la page d'accueil de l'application.

6.2.4 - Fichier faces.config.xml

Le plus simple est de placer ce fichier dans le repertoire WEB-INF de l'application web. Il est aussi possible de préciser son emplacement dans un paramètre de contexte nommé javax.faces.application.CONFIG_FILES dans le fichier web.xml. Il est possible par ce biais de découper le fichier de configuration en plusieurs morceaux. On précise chacun des fichiers séparés par une virgule dans le tag <param-value>, ce qui donne :

```
<context-param>
 <param-name>javax.faces.application.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/ma-faces-config.xml,
                 /WEB-INF/navigation-faces.xml,
                /WEB-INF/beans-faces.xml
    </param-value>
</context-param>
```

Le fichier config-faces.xml permet de définir et de fournir des valeurs d'initialisation pour des ressources nécessaires à l'application utilisant JSF. Il est impératif de respecter DTD proposé par les spécifications JSF à savoir : **`http://java.sun.com/dtd/web-facesconfig_1_0.dtd`** et la racine du document est le tag <face-config>.

```
<? xml version='1.0' encoding='UTF-8'?>
<faces-config
          xmlns="http://java.sun.com/xml/ns/javaee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
          http://java.sun.com/xml/ns/javaee/web-facesconfig_2_1.xsd"
           version="2.1">
</faces-config>
```

6.3 - Ressources

Plusieurs ressources interviennent pour la réalisation de l'application Pizzeria OC. La plupart de ces ressources nous sont fournie par le Framework utilisé, le JSF. Nous citerons entre autres :

- **Les composants graphiques** : ils permettent de la réalisation des interfaces graphiques de l'application web responsive en appliquant les règles web design.
- **La base de données**. Elle permet le stockage des données, données relatives aux catalogues de pizza, les ingrédients en stock, les ingrédients utilisés, les aides mémoires et autres.
- **Le système bancaire** : Il permet divers mode de payement intégré à l'application.

6.4 - Procédure de packaging / livraison

L'application Pizzeria OC étant une application web, elle sera rendue sous forme d'un fichier .war, notamment `pizzeriaoc.war` pour être déployé sur un serveur web, de préférence sur un serveur Tomcat.

Dans le fichier de configuration pom.xml de l'application pizzeriaoc, on aura donc :

```
<project
xmlns=http://maven.apache.org/POM/4.0.0
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>ch.enyo.openclassroom</groupId>
      <artifactId>pizzeriaoc</artifactId>
      <packaging>war</packaging>
      <version>1.0-SNAPSHOT</version>
      <name>Pizzeria OC Maven Webapp</name>
      <url>http://maven.apache.org</url>

      <properties>
            <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
.....
.....
</project>
```

7 - GLOSSAIRE
