|  |
| --- |
| **OC Pizza**  **Projet 8**  Dossier d'exploitation  Version 2 |
| **Auteur**  Alexandre Lardon  *Analyste-Programmeur* |

Table des matières

1 - Versions 4

2 - Introduction 5

2.1 - Objet du document 5

2.2 - Références 5

3 - Pré-requis 6

3.1 - Système 6

3.1.1 - Serveur de Base de données 6

3.1.1.1 - Caractéristiques techniques 6

3.1.2 - Serveur Web 6

3.1.2.1 - Caractéristiques techniques 6

3.1.3 - Serveur de Batches 6

3.1.4 - Serveur de Fichiers 6

3.2 - Bases de données 6

3.3 - Web-services 6

4 - Procédure de Déploiement 8

4.1 - Déploiement de l’application web 8

4.1.1 - Artefacts 8

4.1.2 - Environnement de l’application web 8

4.1.2.1 - Java JDK 8

4.1.2.2 - Serveur Tomcat 8

4.1.2.3 - Sécurisation du serveur grâce au protocole SSL 8

4.1.3 - Microservice 10

4.1.4 - Base de données 10

4.1.4.1 - Power Architect 10

4.1.4.2 - Création Base de données 11

4.1.5 - Feign 13

4.1.6 - JPA / Hibernate / JDBC 13

4.1.7 - Ressources 14

4.1.8 - Vérifications 15

4.2 - Déploiement des Batches 15

4.2.1 - Artefacts 15

4.2.2 - Configuration 15

4.2.2.1 - Fichier Log4j2.xml : 15

4.2.2.2 - Fichier application.properties : 15

4.2.3 - Envoi des mails 15

4.2.4 - Ressources 16

4.2.5 - Vérifications 16

5 - Procédure de démarrage / arrêt 17

5.1 - Base de données 17

5.2 - Batches 17

5.3 - Application web 18

6 - Procédure de mise à jour 19

6.1 - Base de données 19

6.2 - Batches 19

6.3 - Application web 19

7 - Supervision/Monitoring 20

7.1 - Supervision de l’application web 20

8 - Procédure de sauvegarde et restauration 21

8.1 - GitHub 21

8.2 - Base de données 21

9 - Glossaire 23

# Versions

|  |  |  |  |
| --- | --- | --- | --- |
| Auteur | Date | Description | Version |
| Alexandre | 19/10/2020 | Création du document | 0.1.0 |
| Alexandre | 20/10/2020 | Ajout des 5 premiers chapitres | 0.1.0 |
| Alexandre | 22/10/2020 | Document terminé | 1.0 |
| Alexandre | 23/10/2020 | Relecture et quelque modifications apportées | 1.0 |
| Alexandre | 23/10/2020 | Ajout de microservice et mise à jour des versions | 1.1 |

# Introduction

## Objet du document

Le présent document constitue le dossier d’exploitation de l'application OC Pizza

Objectif du document est de spécifier les différentes informations nécessaires à l’exploitation en règle du système et au déploiement de l’application.

## Références

Pour de plus amples informations, se référer :

1. **[DCT – OC Pizza](Projet%20OC%20Pizza%20-%20Dossier%20de%20conception%20technique.docx)** : Dossier de conception technique de l'application
2. **[DCF – OC Pizza](Projet%20OC%20Pizza%20-%20Dossier%20de%20conception%20fonctionnelle%20.docx)** : Dossier de conception fonctionnelle de l’application
3. [**PV – OC Pizza**](Projet%20OC%20Pizza%20-%20PV%20Livraison.docx) : PV de livraison de l’application

# Pré-requis

## Système

### Serveur de Base de données

Serveur de la base de données Postgresql hébergeant le schémas projet8\_OCPizza.

#### Caractéristiques techniques

Postgresql : Version 12.2-2

### Serveur Web

Serveur physique Tomcat hébergeant l'application web.

#### Caractéristiques techniques

Apache Tomcat : Version 9.0.33

### Serveur de Batch

Serveur physique Tomcat 9.0.33 hébergeant le système des Batchs.

### Serveur de Fichiers

Serveur Tomcat 9.0.33 hébergeant les différents fichiers de l’application web.

## Bases de données

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

* **Projet8\_OCPizza :** version 1.0
* **Nombre de table** : 20 tables
* **Interface** : PgAdmin Version 4.19

## Web-services

Les web services suivants doivent être accessibles et à jour :

* **Microservice Interface** : version 1.5
* **Microservice Authentification :** version 1.0
* **Microservice Gestion Utilisateur :** version 1.1
* **Microservice Gestion Commande :** version 1.1
* **Microservice Gestion Produit :** version 1.3
* **Microservice Gestion Batch :** version 1.0

# Procédure de Déploiement

## Déploiement de l’application web

### Artefacts

L’application web est contenu dans le fichier OCPizza\_application.war. Le serveur tomcat déploiera alors l’application à partir du fichier war. JDK grâce à ses bibliothèques et son compileur permets de transformer le code destiné à la machine virtuelle Java. Les données de la base seront lues par l’intermédiaire du driver JDBC permettant l’accès à tomcat.

### Environnement de l’application web

#### Java JDK

Tout d’abord nous devons installer la version 8 de Java JDK. Pour cela, nous nous rendons sur [le site d’Oracle](https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html) et nous téléchargeons la version souhaitée.

Nous nous assurons que la variable d'environnement Java est correctement configurée :

1. Localisons notre répertoire d'installation de Java et copiez le chemin d'accès. Le chemin par défaut est C:\Program Files\Java\jdk <numéro de version>.

2. Depuis le menu Démarrer de Windows, accédons à Paramètres système avancés. La manière d'accéder à ces propriétés varie en fonction de votre version de Windows. Par exemple, sous Windows 10, recherchez Variables d'environnement, puis sélectionnez Modifier les variables d'environnement système.

3. Cliquons sur Variables d'environnement.

4. Dans la section Variables système, cliquons sur Nouvelle.

5. Dans le champ Nom de la variable, spécifions JAVA\_HOME.

6. Dans le champ Valeur de la variable, spécifions le chemin d'accès à notre installation Java, comme défini à l'étape a.

7. Cliquez sur OK.

#### Serveur Tomcat

En premier, nous devons installer Tomcat sur notre système. Téléchargeons la version souhaitée [dans ce lien](https://tomcat.apache.org/download-90.cgi).

Ensuite suivons le protocole d’installation, [à partir de ce lien](https://techexpert.tips/fr/apache-tomcat-fr/installation-de-tomcat-sur-windows/).

Une fois le serveur installé, il faut permettre au serveur d’accéder à la base de données, à l’intérieur du dossier Tomcat, ouvrons le dossier lib puis ajoutons le fichier postgresql-42.2.1.jar que nous trouvons [à cette adresse](https://jdbc.postgresql.org/download.html) dans la colonne intitulée ‘JDBC 42’.

#### Sécurisation du serveur grâce au protocole SSL

Pour sécuriser le serveur, nous devons nous assurer de la confidentialité des données échangées entre l’IHM et le serveur.

Pour réaliser cela, nous devons :

**1 –** Tout d’abord obtenir un certificat, qui sera généré à l’aide de keytool, un executable que l’on retrouve dans le dossier ‘bin’ de notre JDK 1.8.

Pour générer ce fichier il faut entrer une ligne de commande dans votre console :

« keytool -genkey -alias + nom de votre alias + -keyalg RSA -keystore + chemin ou nous souhaitons que le fichier soit créé + nom de votre fichier »

Exemple : keytool -genkey -alias alexCertificat -keyalg RSA -keystore D:\homework \env\tomcat\nomCertif

Ensuite entrons un mot de passe et nous répondons aux questions qui nous sont posées.

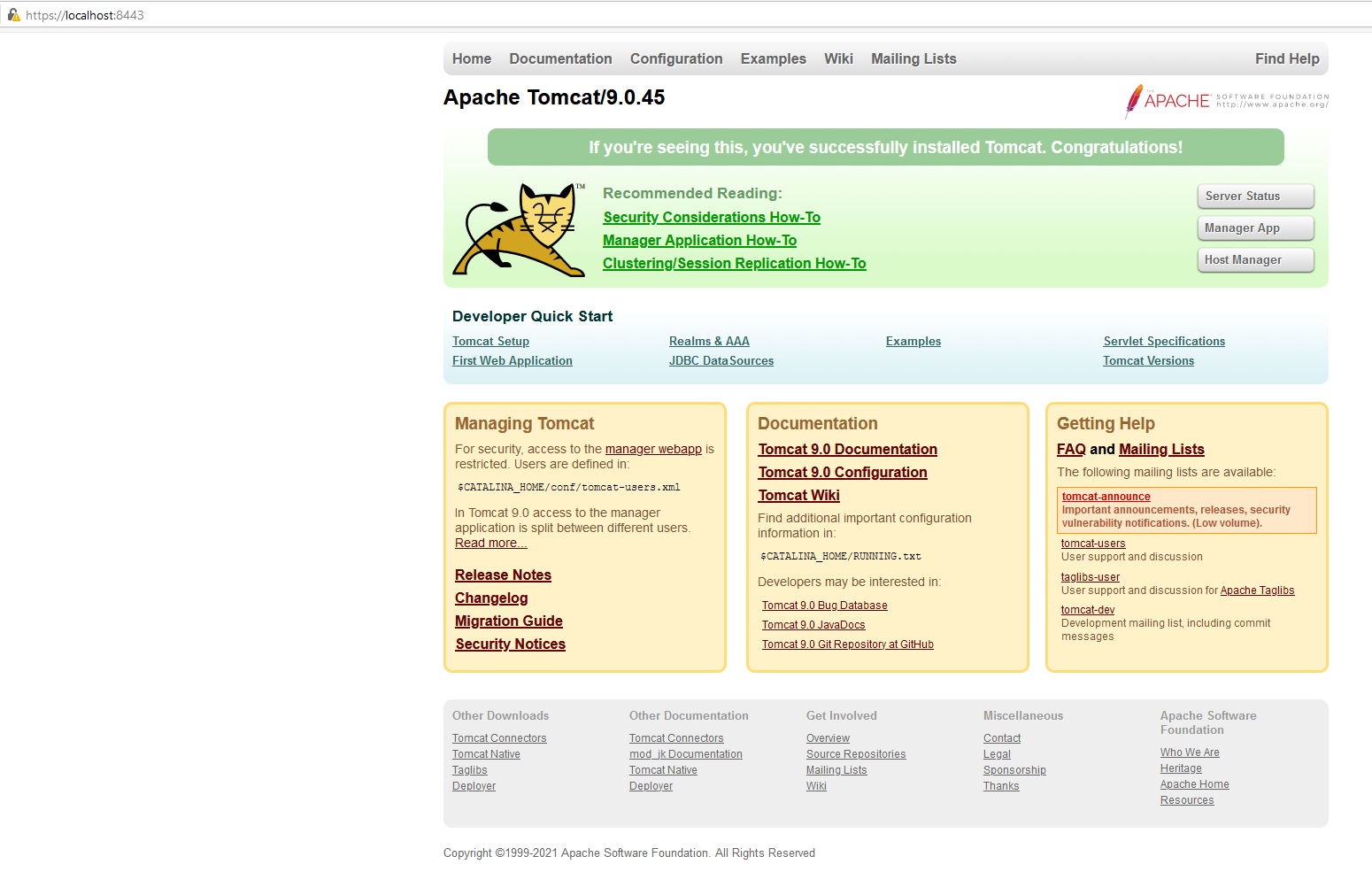
Le fichier sera généré à la fin à l’emplacement que nous avons indiqué.

**2 –** Ensuite nous devons configurer le serveur. Pour cela allez dans le dossier ‘conf’ de tomcat et ouvrons le fichier ‘serveur.xml’, puis modifions le fichier comme indiqué ci-dessous en remplaçant la partie concernant à l’intérieur de la balise <Connector> :

<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"  
 maxThreads="150" scheme="https" secure="true"  
 clientAuth="false" sslProtocol="TLS"  
 keystoreFile="indiquer le chemin de l’emplacement de votre certificat"  
 keystorePass="entrez votre mot de passe" />

**3 –** Pour savoir si cela fonctionne, entrons cette adresse dans le navigateur : <https://localhost:8443/>

Une page web Apache Tomcat 9.0.45 doit s’afficher si la procédure a fonctionné.



### Microservice

Les microservices sont contenus dans des fichiers WAR et le serveur les déploie automatiquement.

Pour cela, nous devons placer les fichiers WAR dans le dossier ‘webapps’ du serveur.

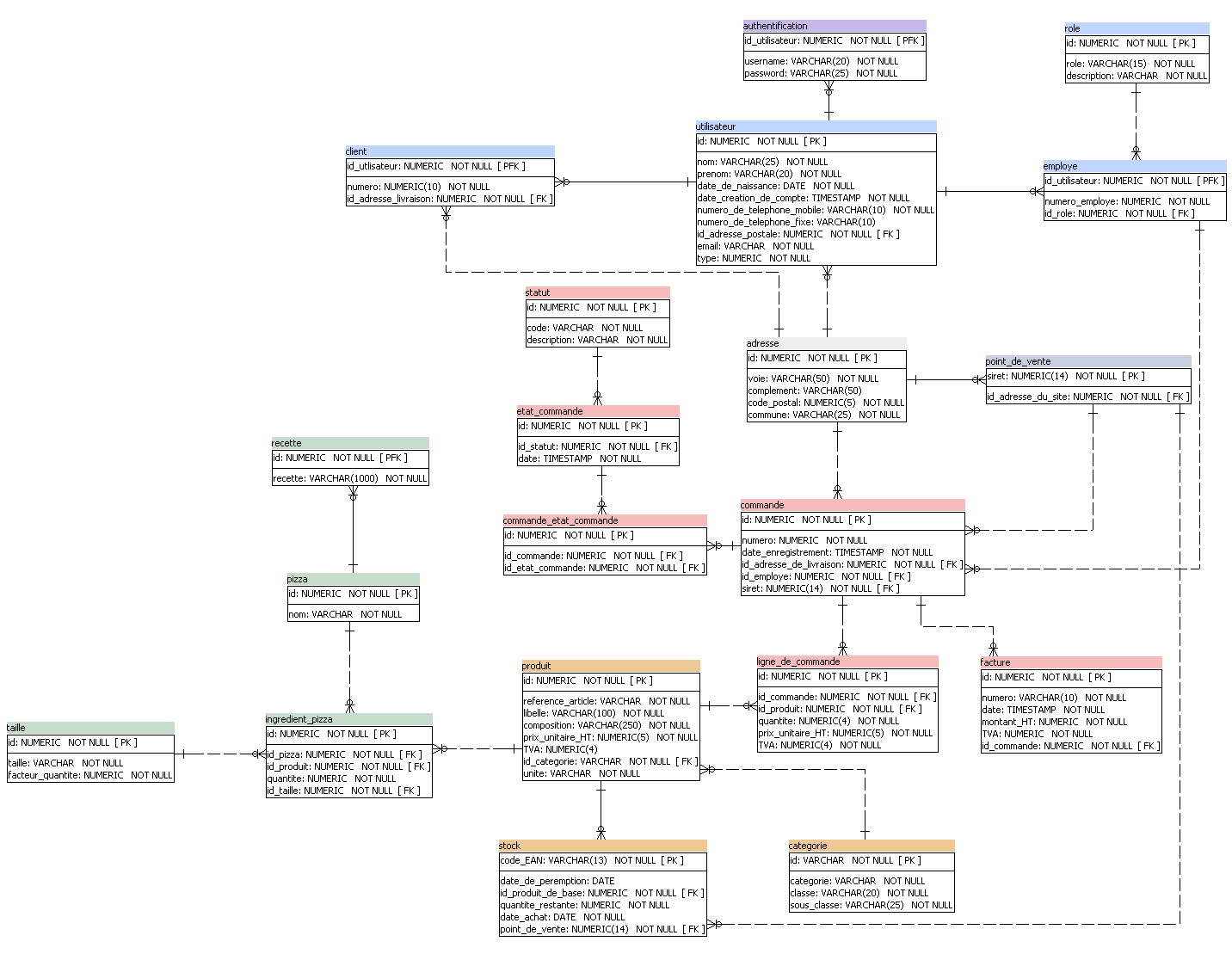
Les dossiers contenants les microservices sont générés automatiquement dans le même dossier.

### Base de données

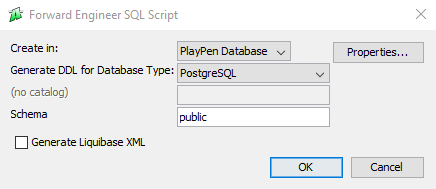
#### Power Architect

Nous devons tout d’abord à partir de notre [modèle physique de données](file:///C:\Users\alexa\Desktop\Soutenance%208\Projet%20OC%20Pizza%20-%20Dossier%20de%20conception%20fonctionnelle%20.docx#Modèle_physique_de_données), créer la structure de la base de données.

Pour cela nous reproduisons dans le logiciel ‘Power Architect ‘ que nous [téléchargeons ici](http://www.bestofbi.com/page/architect_download_os), le modèle physique de données.



Ensuite nous cliquons sur l’icône ‘SQL’ et remplissons comme sur l’image ci-dessous :



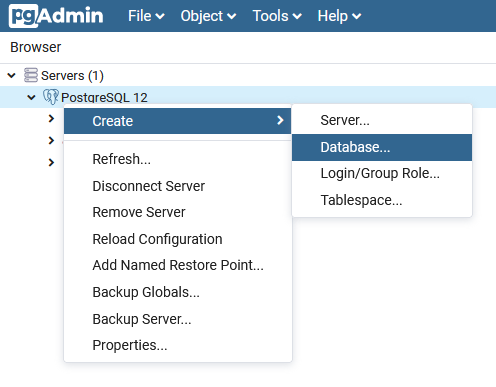
Power Architect va ensuite générer la requête Postgresql permettant la création de la structure de la base de données, faisons une copie de la requête.

#### Création Base de données

**1 -** Téléchargeons tout d’abords les logiciels dont nous avons besoin. Pour cela rendons-nous à cette [adresse](https://www.pgadmin.org/download/pgadmin-4-windows/), puis sélectionnons notre système et la version 4 de pgAdmin.

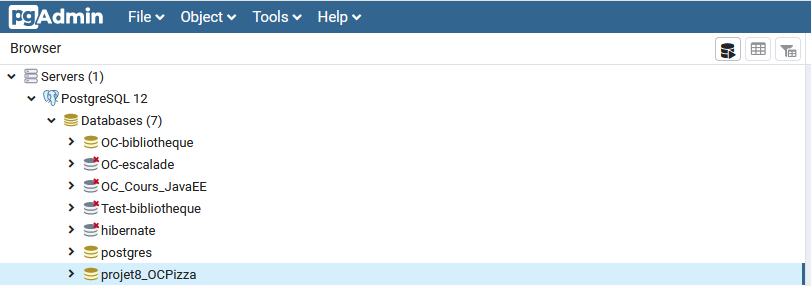
Il installe l’interface pgAdmin et le système de gestion de base de données Postgresql.

**2 -** Rendons-nous sur l’interface de pgAdmin puis créons la base de données.



Il ne reste plus qu’à lui donner un nom et valider.

**3 -** Maintenant que la base est créée, dans la colonne de gauche, sélectionnons notre base de données et cliquons sur l’icone en haut à droite de l’image ci-dessous :



Ensuite il ne reste plus qu’à coller la requête générée par Power Architect, sélectionnons la requête et exécutons.

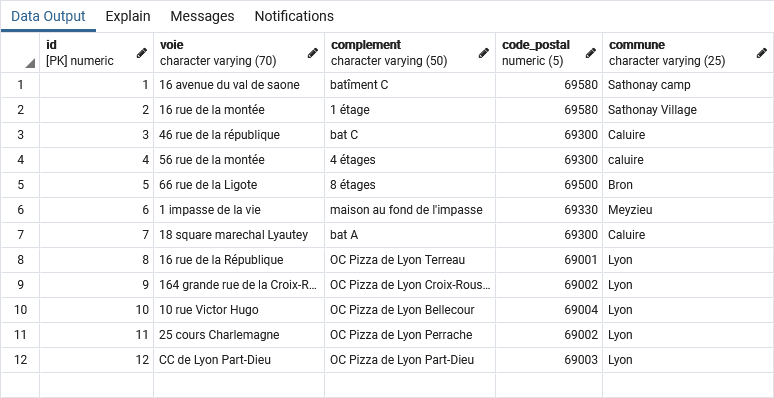
Nous avons toutes les tables et notre structure dans la base de données.



**4 –** Ajoutons les datas dans notre base de données. Pour cela récupérons la requête contenue dans [ce fichier](Sauvegarde_Schema_Données/OCpizza_data.sql).

Puis, comme pour la structure collons la requête dans l’interface pgAdmin, sélectionnons le tout, puis exécutons.

Nous avons les données de toute la base désormais accessible.



### Feign

Feign est un client http qui permet de faire communiquer les microservices entre eux par le biais de requêtes http.

Pour que feign puisse fonctionner il faut :

* Indiquer les noms des microservices dans les fichiers application.properties, pour qu’il puisse les connaitre.
* Indiquer les packages à scanner.
* Indiquer les informations du microservices que l’on souhaite appeler.

Une fois ces informations entrées, Feign est fonctionnel.

### JPA / Hibernate / JDBC

Pour se connecter avec la base de données le logiciel utilise JDBC, cela permet d’avoir accès à la base de données.

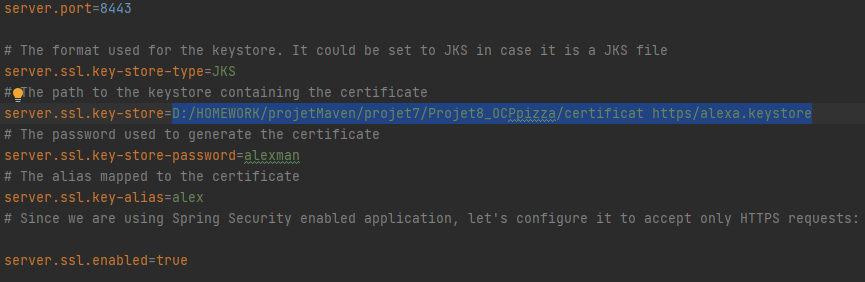
Pour le traitement des données et la récupération de celles-ci dans nos objets, on utilise JPA pour la gestion de la persistance.

Et pour la mise en application, c’est l’ORM Hibernate qui permet de gérer la récupération des données et de les implémenter dans les objets, le tout permettant d’utiliser les opérations correspondant au CRUD.

### Ressources

**configurationHttps.properties** : fichier de configuration du protocole Https situé dans le microservice interface.

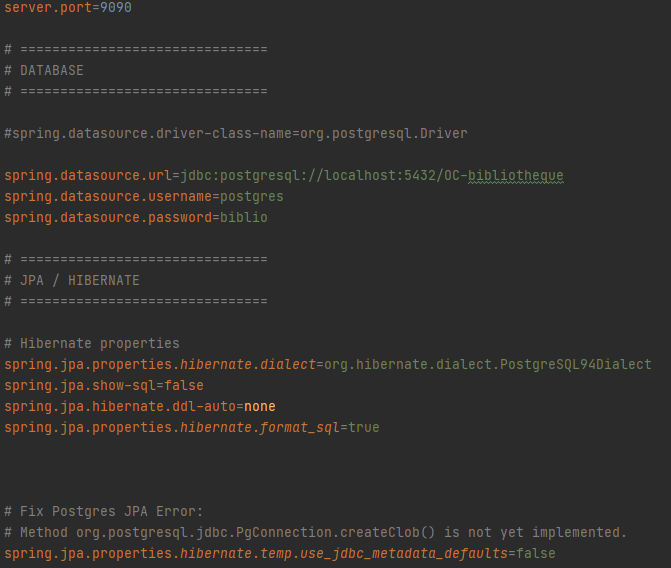
Le chemin d’accès au certificat doit être modifié, s’il doit être déplacé. Il apparait en bleu sur l’image ci-dessous.



Le fichier configurationHttps.properties permet de configurer l’accès de l’application au certificat JKS en définissant l’emplacement et les identifiants du certificat.

**application.properties :** Les accès, à la base de données de chaque microservice doivent être configurer à l'aide des fichiers application.properties.

A l’intérieur de ces fichiers nous retrouvons les informations liées à la connexion à la base de données grâce à JDBC et aux différents paramètres pour le bon fonctionnement de JPA et de Hibernate.



### Vérifications

Afin de vérifier le bon déploiement de l’application, effectuons la mise en route de tous les microservices.

Pour cela, effectuons la mise en route du serveur.

Il suffit de se rendre dans le dossier bin du serveur et de double cliquer sur le fichier ‘startup.bat’.

Le serveur va automatiquement démarrer tous les microservices.

Ensuite rendons nous à l’adresse suivante : <https://localhost:8443/Accueil>

Il ne reste plus qu’à vérifier le bon fonctionnement de l’application web.

## Déploiement des Batchs

### Artefacts

Les batchs de l’application OC Pizza sont automatisés. L’envoi d’un batch survient à la suite de l’action d’un employé lorsque :

* La commande est validée via l’application.
* La commande est prête à être retirée.
* La commande a été livrée (envoi facture).
* Le niveau de stock d’un produit est bas.

### Configuration

Voici les différents fichiers de configuration :

* **Log4j2.xml** : fichier de configuration des logs.
* **Application.properties** : fichier de configuration de la base de données et de spring data JPA / Hibernate, et des variables d’environnement comme vu ci-dessus.

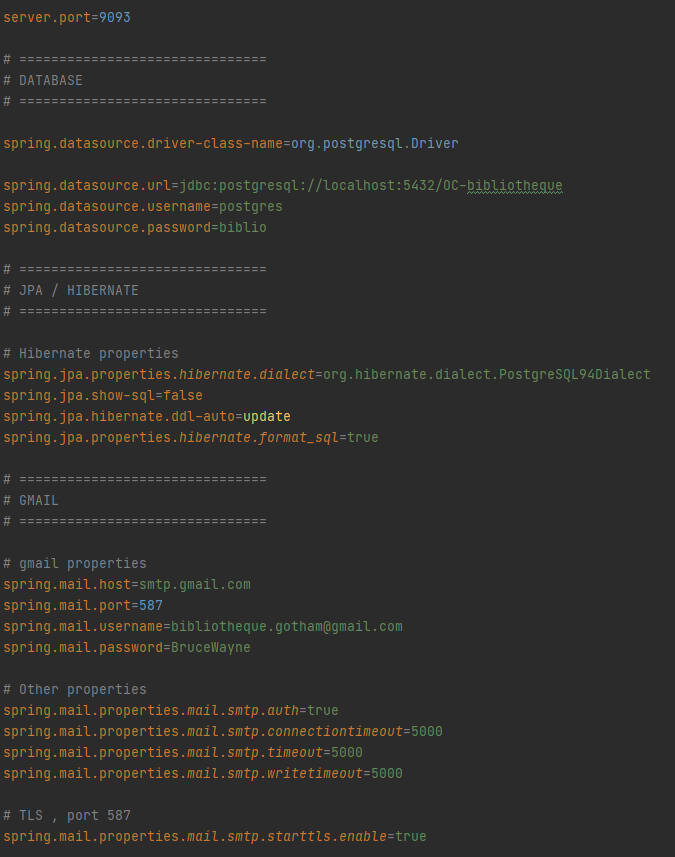
#### Fichier Log4j2.xml :

Le fichier log4j2.xml sert à paramétrer la gestion des logs pour log4j2(structuration de l’erreur, niveau de déclenchement de l’erreur…)

#### Fichier application.properties :

Le fichiers application.properties se situe dans le package ressource. Nous retrouvons dans le fichier, les identifiants d’accès à la base de données ainsi que les paramètres servant à configurer Spring Data JPA et Hibernate.

Nous retrouvons également la configuration de l’adresse mail permettant l’envois des emails.



### Envoi des mails

Les mails sont écrits et contenus dans des fichiers HTML.

FreeMarker génère un email à partir du fichier html permettant l’envoi du mail via SMTP.

### Ressources

* **Serveur** : Tomcat version 9.0.33
* **Base de données** : Postgresql 12.2-2
* **Génération eMail**: FreeMarker 2.3.30

### Vérifications

Afin de vérifier le bon déploiement des batchs, il faut simuler une commande.

1. Pour cela, configurons la variable d’environnement ‘Test = *true’ qui se trouve dans le fichier application.properties.*
2. Connectez-vous à un compte.
3. Ensuite, commander une ou plusieurs pizzas, choisissez le mode de livraison, puis validez la commande.
4. Vérifiez que vous avez bien reçu l’email de confirmation de la commande.
5. Vérifiez que vous avez bien reçu l’email de retrait en point de vente ou celui détaillant la livraison à votre domicile avec la facture.

# Procédure de démarrage / arrêt

## Base de données

Une fois les identifiants et l’adresse saisie dans les fichiers application.properties, l’accès à la base de données est automatique.

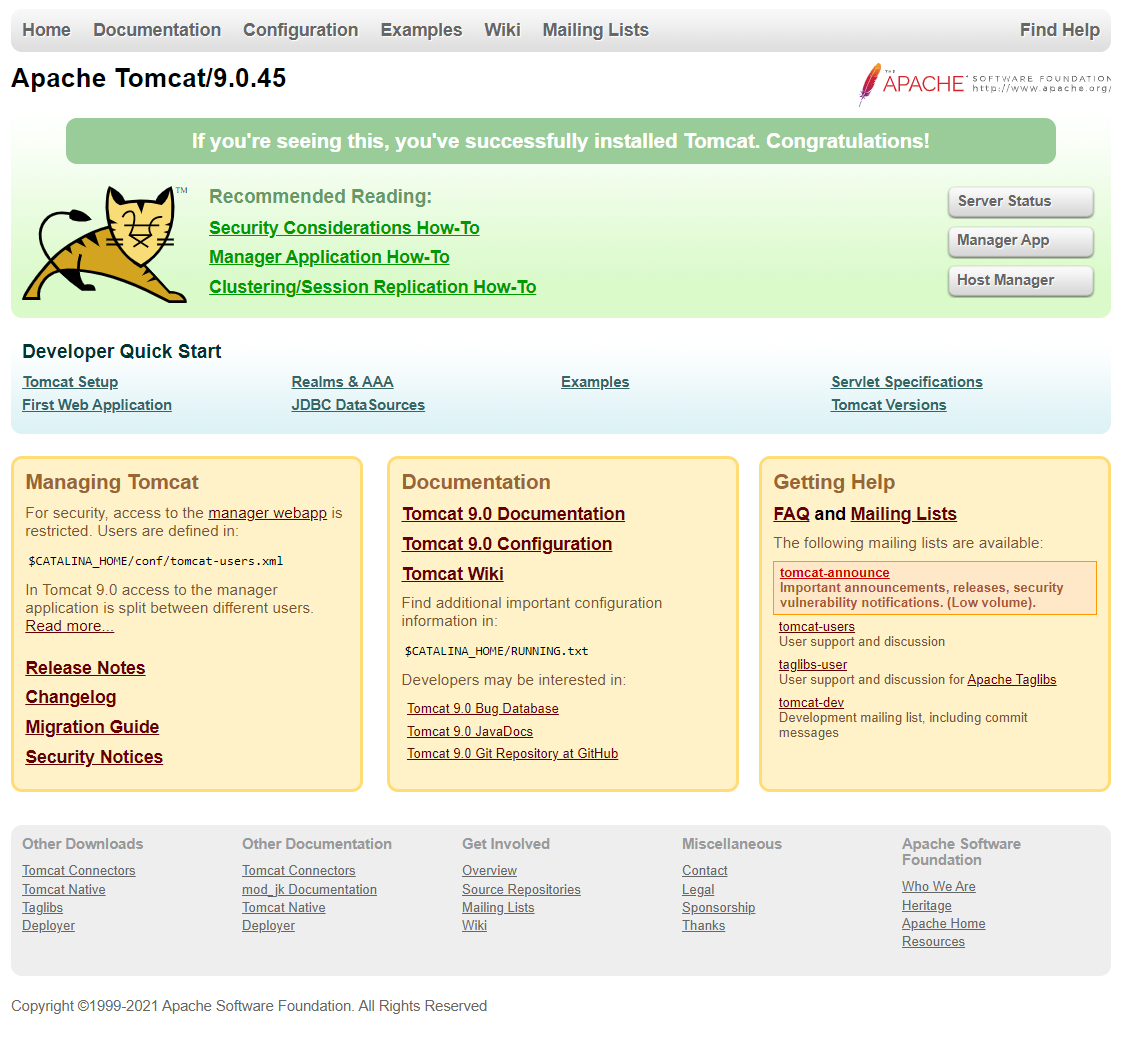
L’accès à la base de données se fait par l’intermédiaire de l’interface pgAdmin 4, en saisissant les informations d’authentification demandées.

## Batchs

Les batchs sont automatiques pour le démarrage ou l’arrêt du microservice batch, nous devons passer par le serveur pour l’arrêter ou le lancer.

- Pour cela entrons dans notre navigateur l’adresse suivante : <https://localhost:8443>.

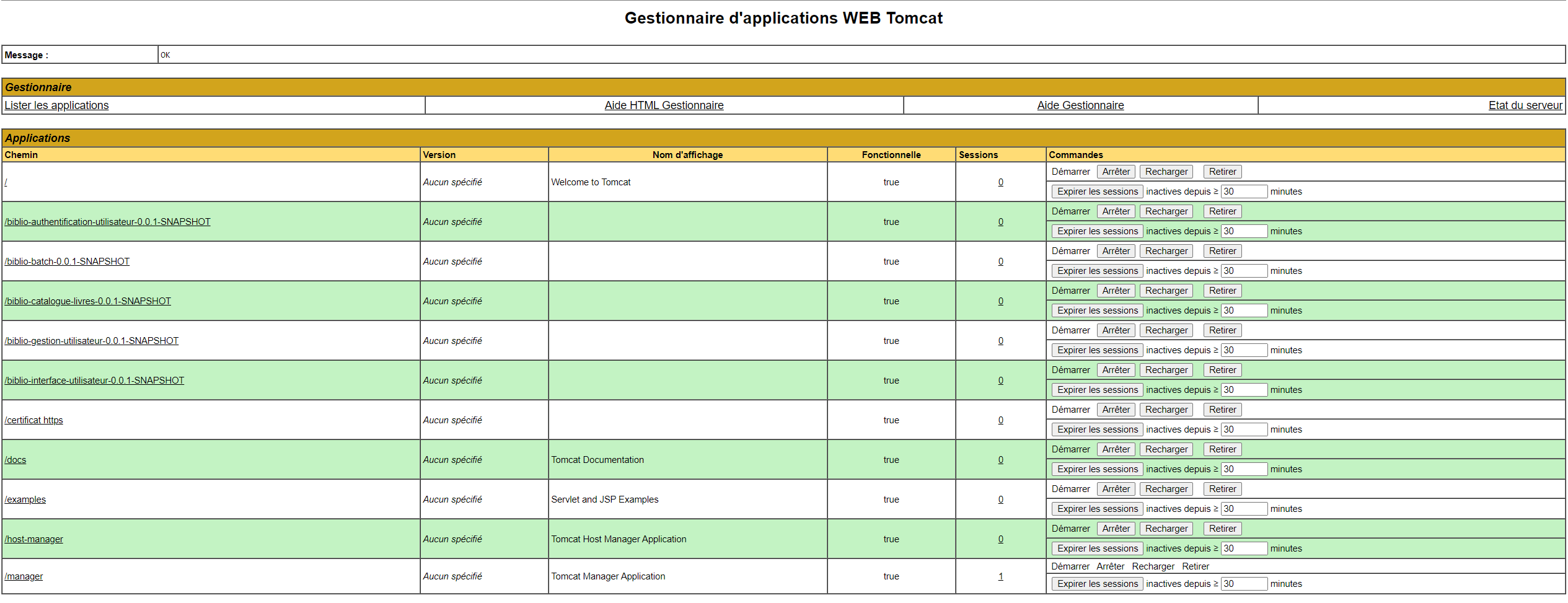
- Puis sélectionnons ‘Manager App’ :



- Entrons ensuite les identifiants de connexion demandés.

- Nous sommes dans le gestionnaires des applications se trouvant sur le serveur.

- Nous pouvons choisir de démarrer ou arrêter le service des Batchs.



## Application web

L’application web doit être démarrées ou arrêtées par l’intermédiaire du serveur complet ou des différents microservices comme indiqué ci-dessus.

# Procédure de mise à jour

## Base de données

La mise à jour de la version de la base de données est mise à jour par le biais du fichier pom.xml dans chacun des microservices ayant accès à la base de données.

Il suffit de modifier la version de la dépendance avec la version souhaitée dans le fichier pom des microservices de l’application.

<dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <version>42.2.12</version>  
</dependency>

## Batchs

Les batchs peuvent être mises à jour en modifiant les fichiers html, se trouvant à l’intérieur du dossier ‘resources/templates’ du microservices batch.

## Application web

L’application web est mise à jour par l’intermédiaire des différents microservices.

# Supervision/Monitoring

## Supervision de l’application web

Afin de tester que l’application web est toujours fonctionnelles, exécutons à partir de notre ide les tests automatisés et assurons-nous qu’ils soient tous vert.

Sinon à partir d’une console, entrons le chemin du dossier, puis saisissons la ligne de commande : mvn test

Assurons-nous que le résultat affiché ne comporte aucune erreur.

# Procédure de sauvegarde et restauration

## GitHub

L’ensemble du projet est sauvegardé sur GitHub. Vous pouvez le restaurer à tout moment.

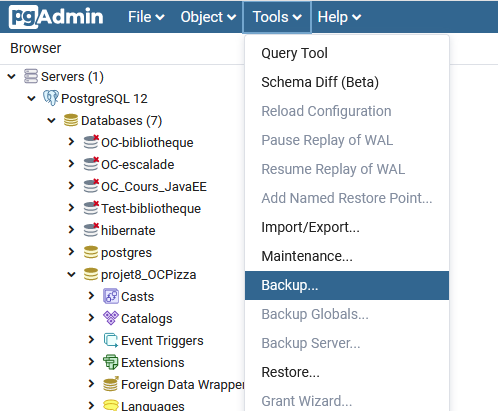
Pour cela, Il suffit de récupérer à cette adresse le repository : [**https://github.com/aquel69/Projet8\_OCPizza**](https://github.com/aquel69/Projet8_OCPizza)

Dans le projet, vous retrouverez les différentes sauvegardes de la structure et des data de la base de données.

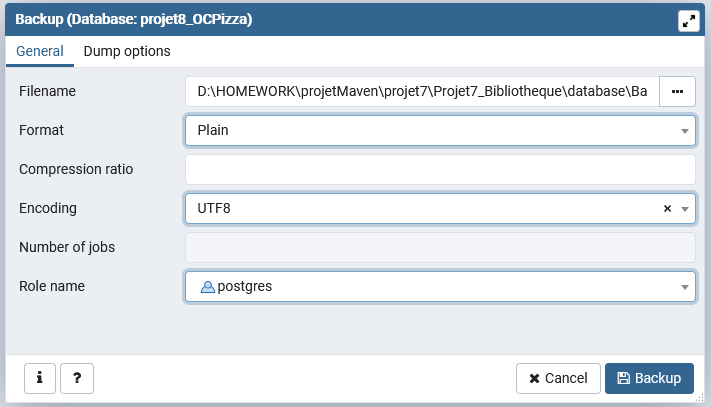
## Base de données

Nous pouvons également sauvegarder les modifications que nous allons apporter à la base de données, suivant la procédure suivante :

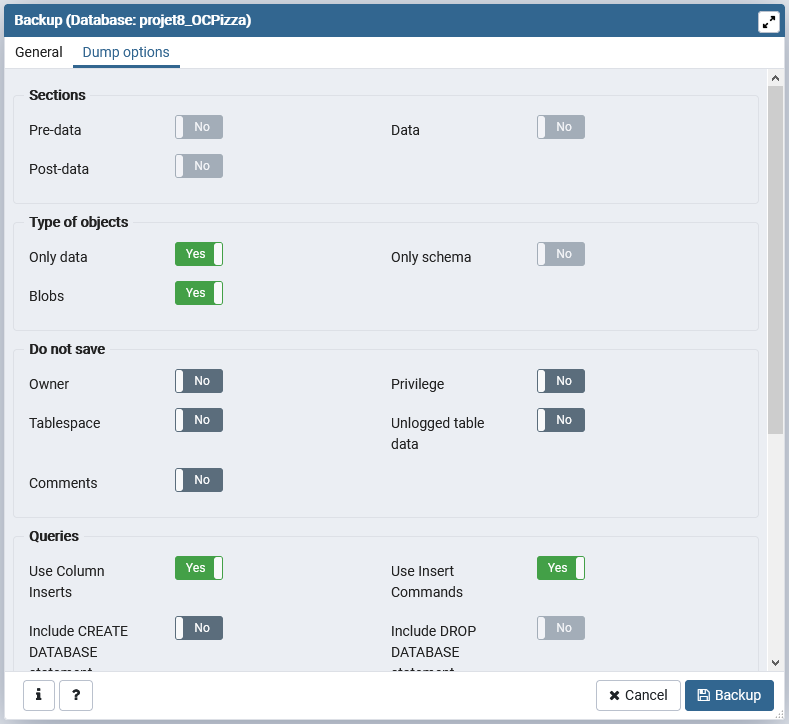
* Par l’intermédiaire de pgAdmin 4, allez dans l’onglet ‘Tools’ et sélectionnez ‘Backup’.



* Ensuite remplissons les différents champs du premier onglet, le champ ‘Filename’ est le chemin de votre sauvegarde plus le nom de votre fichier, pour le reste remplissons selon l’exemple ci-dessous :



* Dans le deuxième onglet, modifions les boutons coulissant comme sur l’exemple ci-dessous :



Vérifions dans le dossier correspondant au chemin que nous avons entré si le fichier de sauvegarde a bien été créé.

# Glossaire

|  |  |
| --- | --- |
| **Log4j2** | Log4J2 est une API de log proposée par la fondation Apache, pour la gestion des logs. |
| **Postgresql** | PostgreSQL est un système de gestion de base de données relationnelle et objet permettant la gestion de la base de données. |
| **Maven** | Maven est un outil de gestion et d’automatisation de production des projets logiciels Java en général et JavaEE en particulier. |
| **Tomcat** | Serveur permettant d’exécuter des applications Web développées avec les technologies Java. |
| **Power Architect** | Logiciel qui permet de modéliser une base de données puis de générer automatiquement le schéma dans le système de gestion de base de données de notre choix. |
| **Hibernate** | Hibernate est un ORM (Object Relation Mapping). C’est un framework permettant d’implémenter les interfaces d’accès aux données afin de les stocker dans des objets. |
| **IHM** | IHM (Interface-Homme-Machine) sont les moyens et outils mis en œuvre afin qu'un humain puisse contrôler et communiquer avec une machine. |
| **JDBC** | JDBC (Java DataBase Connectivity) est une API Java constituée d'un ensemble d'interfaces et de classes permettant de se connecter à n’importe quelle base de données et d’interagir avec elle. |
| **JPA** | JPA **(**Java Persistence API) Interface Java, définissant un certain nombre de mots-clés et de normes à respecter. Elle est utilisée pour accéder, gérer et conserver les données entre l'objet Java et la base de données relationnelle. |
| **JDBC** | JDBC (Java DataBase Connectivity) est une API Java constituée d'un ensemble d'interfaces et de classes permettant de se connecter à n’importe quelle base de données et d’interagir avec elle. |
| **API** | API (Application Programming Interface) Interface permettant à un logiciel d’accéder aux services d’un autre logiciel par le biais d’un ensemble normalisé de classes, de méthodes, de fonctions et de constantes. |
| **SMTP** | SMTP (Simple Mail Transfer Protocol) protocole de communication utilisé pour transférer le courrier électronique. |
| **HTTPS** | HTTPS(HyperText Transfer Protocole Secure) Protocole permettant de sécuriser les données échangées entre le client et le serveur. |
| **HTML** | HTML(Hyper Text Markup Language) Langage de balisage conçu pour représenter les pages web. |
| **JDK** | Le JDK (Java Development Kit) désigne un ensemble de bibliothèques logicelles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé, transformé en bytecode destiné à la machine virtuelle Java. |
| **SSL** | Le SSL créé un canal sécurisé entre deux machines ou appareils communiquant sur Internet ou un réseau interne. Son usage le plus courant est la sécurisation de la communication entre un navigateur web et un serveur web. |
| **WAR** | Le fichier WAR est une archive Java. Elle permet de structurer et de paramétrer l’application en elle-même. Il détermine également les dépendances et les variables lors du déploiement d’une application web. |
| **GitHub** | GitHub est un service web d’hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. |
| **FreeMarker** | **FreeMarker** est un moteur de template, il est fourni par **Apache** comme une bibliothèque **Java** source ouverte. **FreeMarker** lit des fichiers modèle et les combine avec des objets **Java** pour générer un texte de sortie (Html, email, ..). |