|  |
| --- |
| **OC PIZZA**  **Projet 8 : Documentez votre système de gestion de pizzeria**  Dossier d'exploitation  Version 1.0 |
| **Auteur**  AUBRUN Éric  *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

3.2 - Bases de données 6

3.3 - Web-services 6

3.3.1 - PAYMILL, API de paiement en ligne intégrant des bibliothèques Java, voici la procédure d’activation : 6

3.3.2 - JDBC (Java DataBase Connectivity) API d’accès aux bases de données relationnelles : 6

3.3.3 - SMS API d’envoi de SMS en java via un SDK 7

3.3.4 - JMM : API de géolocalisation « Java MaxMind » 7

3.3.5 - JAAS (Java Authentication and Authorization Service) : API de gestion de l’authentification. 8

3.3.6 - JavaMail API pour la gestion de courrier électronique. 8

4 - Procédure de déploiement 9

4.1 - Déploiement de l’application web 9

4.1.1 - Composition de l’application web 9

4.1.2 - Variables d'environnement 9

4.1.3 - Configuration 10

4.1.4 - Déploiement 10

4.1.4.1 - Création de l’application : 10

4.1.4.2 - Configuration des variables d’environnement : 10

4.1.4.3 - Envoyer l’application sur le serveur 11

4.1.5 - Vérifications 11

4.2 - Déploiement de la base de données 11

4.2.1 - Lancement des migrations 11

4.2.2 - Création d’un superutilisateur 11

4.2.3 - Import des données 12

4.2.4 - Vérifications 12

5 - Procédure de démarrage / arrêt 13

5.1 - Démarrage / arrêt sur le site heroku.com 13

5.2 - Démarrage / arrêt en ligne de commande 13

6 - Procédure de mise à jour 14

7 - Supervision/Monitoring 15

8 - Procédure de sauvegarde et restauration 16

8.1 - Sauvegarde de la base de données 16

8.2 - Restauration de la base de données 16

9 - Glossaire 17

# Versions

|  |  |  |  |
| --- | --- | --- | --- |
| Auteur | Date | Description | Version |
| Xxx | JJ/MM/AAAA | Création du document | XXX |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Objet du document

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

Ce document décrit les spécifications techniques du déploiement de cette application.

## Références

Pour de plus amples informations, se référer :

1. **PDOCPizza\_01\_fonctionnelle.pdf – 1.0:** Le Dossier de conception technique de l’application
2. **PDOCPizza\_02\_technique.pdf – 1.0** : Le Dossier de conception technique de l'application
3. **PDOCPizza\_04\_livraison.pdf – 1.0:** Le PV de livraison finale

# Pré-requis

## Système

L’application web OC PIZZA est hébergé sur le serveur OVH.

L’application est liée au nom de domaine : « ocpizza.fr »

## Bases de données

Le SGBD utilisé par l’application est PostgreSQL dans sa version 12.5.

La base de données est hébergée sur le serveur OVH. Pour un accès réactif, on préfèrera une bande passante publique de 500 Mbits/s. Après validation de la commande, il est possible que la durée de livraison du serveur prenne entre 2 et 4 jours.

## Web-services

Le fonctionnement de l’application nécessite que les web-services suivants soit activés et opérationnels :

### PAYMILL, API de paiement en ligne intégrant des bibliothèques Java, voici la procédure d’activation :

1. « *Créez votre propre tunnel de commande ou page de paiement.*
2. *Intégrez notre pont JavaScript afin de récupérer les informations de paiement de manière sécurisée.*
3. *Intégrez PAYMILL à votre serveur soit en utilisant une de nos bibliothèques, soit en appelant notre API directement*. »[[1]](#footnote-1)

### JDBC (Java DataBase Connectivity) API d’accès aux bases de données relationnelles :

Toutes les classes de JDBC sont dans le package java.sql. Il est donc nécessaire de l’importer dans tous les programmes devant l’utiliser (import java.sql. \* ;).

### SMS API d’envoi de SMS en java via un SDK

Exemple de requête :

**public class ExempleClientHttpAPI {**

**private static final String URL = "https://api.smsmode.com/http/1.6/";**

**private static final String PATH\_SEND\_SMS = "sendSMS.do";**

**private static final String PATH\_SEND\_SMS\_BATCH = "sendSMSBatch.do";**

**private static final String ERROR\_FILE = "The specified file does not exist";[[2]](#footnote-2)**

Les 4 variables initialisées au début de ma requête vont permettre de créer des méthodes : une méthode de réception de SMS, une méthode de création de SMS, une méthode batch d’envoi de SMS et une méthode convertToString.

### JMM : API de géolocalisation « Java MaxMind »

L’import de la dépendance suivante dans le POM de l’application permet d’avoir accès aux bibliothèques de l’API :

<dependency>

<groupId>com.maxmind.geoip</groupId>

<artifactId>geoip-api</artifactId>

<version>1.3.1</version>

</dependency>

Il est également nécessaire d’importer les bases de données de pays et villes, propres à l’API Maxmind :

public class MyConstants {

// Country Data.

public static final String DATABASE\_COUNTRY\_PATH = "F:/GeoLite2/GeoLite2-Country.mmdb";

// City Data.

public static final String DATABASE\_CITY\_PATH = "F:/GeoLite2/GeoLite2-City.mmdb";

………

Nous devons ensuite créer une classe comportant un Main afin de charger ces bases de données.

### JAAS (Java Authentication and Authorization Service) : API de gestion de l’authentification.

Pour la mise en place de cette API, nous suggérons le recours à l’excellent « Guide to the Java Authentication and Authorization Service (JAAS) » :

« *When using JAAS in an application, several APIs are involved:*

* *CallbackHandler: Used for gathering user credentials and optionally provided when creating the LoginContext*
* *Configuration: Responsible for loading LoginModule implementations and can be optionally provided at LoginContext creation*
* *LoginModule: Effectively used for authenticating users*

*We'll use the default implementation for the Configuration API and provide our own implementations for the CallbackHandler and the LoginModule APIs*. »[[3]](#footnote-3)

### JavaMail API pour la gestion de courrier électronique.

Il faut créer une classe qui permette le chargement des bibliothèques nécessaires ainsi que la création de divers objets nécessaire à l’implémentation de l’API :

import java.util.Properties;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.PasswordAuthentication;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeMessage;

public class SendEmail {

public static void main(String[] args) {

// Recipient's email ID needs to be mentioned.

String to = "destinationemail@gmail.com";

// Sender's email ID needs to be mentioned

String from = "fromemail@gmail.com";

final String username = "manishaspatil";//change accordingly

final String password = "\*\*\*\*\*\*";//change accordingly

// Assuming you are sending email through relay.jangosmtp.net

String host = "relay.jangosmtp.net";

Properties props = new Properties();

props.put("mail.smtp.auth", "true");

props.put("mail.smtp.starttls.enable", "true");

props.put("mail.smtp.host", host);

props.put("mail.smtp.port", "25");

// Get the Session object.

Session session = Session.getInstance(props,

new javax.mail.Authenticator() {

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication(username, password);

}

});[[4]](#footnote-4)

A la suite de quoi, nous devons instancier les objets précédemment créés afin de constituer le pattern de l’API. Nous pouvons, si nous le souhaitons, injecter cette API dans un batch dans le but d’automatiser l’envoie de mail.

# Procédure de déploiement

## Déploiement de l’application web

### Composition de l’application web

L’application OCPIZZA est construite sous la forme d'une archive ZIP contenant les répertoires suivants :

* OCPIZZA : Contient les fichiers de configuration de Spring Boot et Angular.
* Procfile : Contient les instructions pour démarrer l’application.
* requirements.txt : contient les librairies nécessaires pour que l’application fonctionne.
* Static : Contient les fichiers .CSS et .JS de l’application.
* Templates : Contient les fichiers HTML de l’application.
* Ventes : Contient les fichiers liés au package « Ventes » de l’application.
* Production : Contient les fichiers liés au package « Production » de l’application.
* Docs : Contient la documentation de l’application.

Le fichier Procfile contient les instructions exécutées par heroku pour démarrer l’application. Le contenu de ce fichier doit a minima être le suivant pour que l’application fonctionne :

web: gunicorn pizzapp.run :app

### Variables d'environnement

L’application nécessite la configuration des variables d’environnement suivantes :

|  |  |  |  |
| --- | --- | --- | --- |
| Nom | Valeur | Obligatoire | Description |
| ENV | **PRODUCTION** | Oui | Indique à l’application qu’il faut utiliser la configuration de production et non de développement |
| SECRET\_KEY | **-~]Oe| §;rE[?§/w^$cgmh(4’** | Oui | Nécessaire au démarrage de Django. *Cette clé est la clé de production et ne doit figurer nulle part ailleurs que sur le serveur de production* |

### Configuration

Voici les différents fichiers de configuration :

* procfile : fichier contenant les commandes utilisées par heroku pour le déploiement de l’application.
* requirements.txt : fichier contenant la liste des librairies à installer sur le serveur heroku pour que l’application fonctionne correctement.
* pizzapp/pizzapp/settings.py : fichier de configuration l’application Django.

### Déploiement

Heroku propose différentes méthodes pour déployer une application. Nous conseillons cependant d’effectuer un déploiement en ligne de commande en utilisant « Heroku CLI ».

Les étapes présentées ci-dessous nécessite que « heroku CLI » soit installé, que le projet soit suivi avec « Git » et il faut être sur la branche « master » du projet.

#### Création de l’application :

Une fois « heroku cli » installé la connexion au compte OC-PIZZA effectuée, il faut se rendre à la racine du projet de l’application et entrer la commande suivante :

**$ heroku create pizzapp**

#### Configuration des variables d’environnement :

Les variables d’environnement se définissent via la ligne de commande de la façon suivante :

**$ heroku config : set ENV=’PRODUCTION’**

**$ heroku config : set SECRET\_KEY= ‘\*\*\*\*\*\*\*\*\*\*’**

Pour vérifier que les variables ont bien été configurées, utilisez la commande :

**$ heroku config**

#### Envoyer l’application sur le serveur

Une fois les étapes précédentes réalisées, il suffit d’effectuer un « Push » sur le serveur :

**$ git push heroku master**

### Vérifications

Une fois le déploiement terminé, les lignes suivantes doivent apparaître dans le terminal :

**remote : https://pizzapp.fr/ deployed to Heroku**

**remote: Verifying deploy... done.**

Pour s’assurer du bon déploiement, il convient ensuite de se connecter à l’adresse www.pizzapp.fr afin de vérifier que l’application est bien en ligne.

## Déploiement de la base de données

Le déploiement de la base de données sur la plateforme Heroku nécessite que l’application ait été déployée comme présenté précédemment.

De la même manière que pour le déploiement de l’application, celui de la base de données est présenté ici en utilisant la ligne de commande et « heroku CLI ».

### Lancement des migrations

Avant d’importer les données en elle-même, il est nécessaire de créer et configurer les tables de la base de données via la commande suivante :

**$ heroku run python manage.py migrate**

### Création d’un superutilisateur

Une fois la base de données créée, il faut ajouter un super utilisateur via la commande :

**$heroku run python manage.py createsuperuser**

### Import des données

Il est possible d’importer les données de la version de développement de l’application sur le serveur de production. Pour cela, il faut créer un dump de la base de données de développement, puis l’envoyer l’importer sur le serveur d’heroku.

- Création d’un dump de la base de données :

**$ ./manage.py dumpdata pizzapp > pizzapp/dumps/pizzapp.json**

- Import des données dans la base :

**$ heroku run python manage.py loaddata pizzapp/dumps/pizzapp.json**

### Vérifications

La vérification de l’état de la base de données peut se faire directement via le site de Heroku dans la rubrique « Ressources » du projet.

Il faut ensuite se rendre dans la partie « Add-ons » puis cliquer sur « Heroku Postgres :: Database ».

On retrouve dans cet espace les informations sur la base de données telles que : son état (active ou non), le nombre de ligne ou encore l’espace utilisé.

# Procédure de démarrage / arrêt

La gestion de l’application peut se faire de deux manières, soit depuis le site de Heroku soit en ligne de commande après avoir téléchargé l’interface « heroku-cli » :

## Démarrage / arrêt sur le site heroku.com

Une fois déployée sur Heroku, l’application est par défaut en ligne.

Afin de désactiver l’application il faut se rendre dans la rubrique « Settings », puis changer le statut « Maintenance Mode » à « ON ».

Il suffira ensuite de faire la démarche inverse afin de rétablir l’application.

## Démarrage / arrêt en ligne de commande

Afin de démarrer ou arrêter l’application via la ligne de commande, il faut utiliser les commandes suivantes :

Mise en marche de l’application :

**$ heroku run**

Arrêt de l’application :

**$ heroku ps :scale web=0**

# Procédure de mise à jour

La mise à jour de l’application nécessite de passer celle-ci en mode maintenance.

La mise en mode maintenance peut s’effectuer directement via le tableau de bord heroku (comme présenté dans la section précédente), soit directement en ligne de commande via la commande suivante :

**$ heroku maintenance :on**

Lorsque le mode maintenance est activé, un message s’affiche lors de la connexion à l’application et les utilisateurs n’ont plus accès à celle-ci.

Une fois la mise à jour de l’application effectuée, l’application peut être réactivée comme ceci :

**$ heroku maintenance :off**

# Supervision/Monitoring

La plateforme Heroku fournit des outils de monitoring dans la rubrique « Metrics » de l’application.

Plus d’informations sont disponibles dans la documentation de Heroku, à l’adresse suivante : https://devcenter.heroku.com/categories/monitoring-metrics

Nos recommandations de suivi à mettre en place sont a minima les suivantes :

- Utilisation/Saturation du processeur

- Utilisation/Saturation de la mémoire vive

- Rapprochement de la limite de stockage de la base de données.

# Procédure de sauvegarde et restauration

Le processus de sauvegarde et de restauration de l’application passe essentiellement par la sauvegarde des données de celle-ci. Heroku propose cette fonctionnalité.

## Sauvegarde de la base de données

**$ heroku pg:backups:capture --app pizzapp**

## Restauration de la base de données

**$ heroku pg:backups:restore <nom sauvegarde> DATABASE\_URL --app pizzapp**

# Glossaire

|  |  |
| --- | --- |
| **SGBD** | Système de gestion de base de données |
| **DJANGO** | Framework permettant la réalisation d’application web en Python. |

1. Source : <https://www.paymill.com/fr/integration/web-api/> [↑](#footnote-ref-1)
2. Source : <https://www.smsmode.com/api-sms/?gclid=CjwKCAjw-e2EBhAhEiwAJI5jgyDX9n_jJPfnAWTmniulY1QTY0i_8gT3wQK9awi2BHk4sYfQQarvtRoCRHQQAvD_BwE#1518520975385-9f96526b-74cf> [↑](#footnote-ref-2)
3. Source : <https://www.baeldung.com/java-authentication-authorization-service> [↑](#footnote-ref-3)
4. Source : <https://www.tutorialspoint.com/javamail_api/javamail_api_quick_guide.htm> [↑](#footnote-ref-4)