OC Pizza

IT Consulting & Development

# OC PIZZA

# Système de gestion pour pizzerias

## Dossier de conception technique

## Version 1

**Auteur**
Yann Rouzaud
Analyste - Programmeur

# TABLE DES MATIÈRES

# 1. VERSIONS

| Auteur | Date | Description | Version |
|---|---|---|---|
| Yann Rouzaud | 01/01/2023 | Création du document | 1.0 |
| Yann Rouzaud | 17/01/2023 | Modification du document | 1.1 |
| | | | |
| | | | |

# 2.INTRODUCTION

## 2.1.Objet du document

Le présent document constitue le dossier de conception technique de l'application d'OC PIZZA. Il est destiné au développeurs, mainteneurs et de l'équipe technique du client.

Ce document a pour objectif de présenter les technologies utilisées pour la mise en place du système de gestion.

Les éléments du présents dossiers découlent :

- de la demande initiale du client
- des échanges avec le client
- du dossier de conception fonctionnelle référencé DCT - 001

## 2.2.Références

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

1. **DCT - 001**: Dossier de conception fonctionnelle de l'application
2. **DCT - 003**: Dossier d'exploitation de l'application

# 3. ARCHITECTURE TECHNIQUE

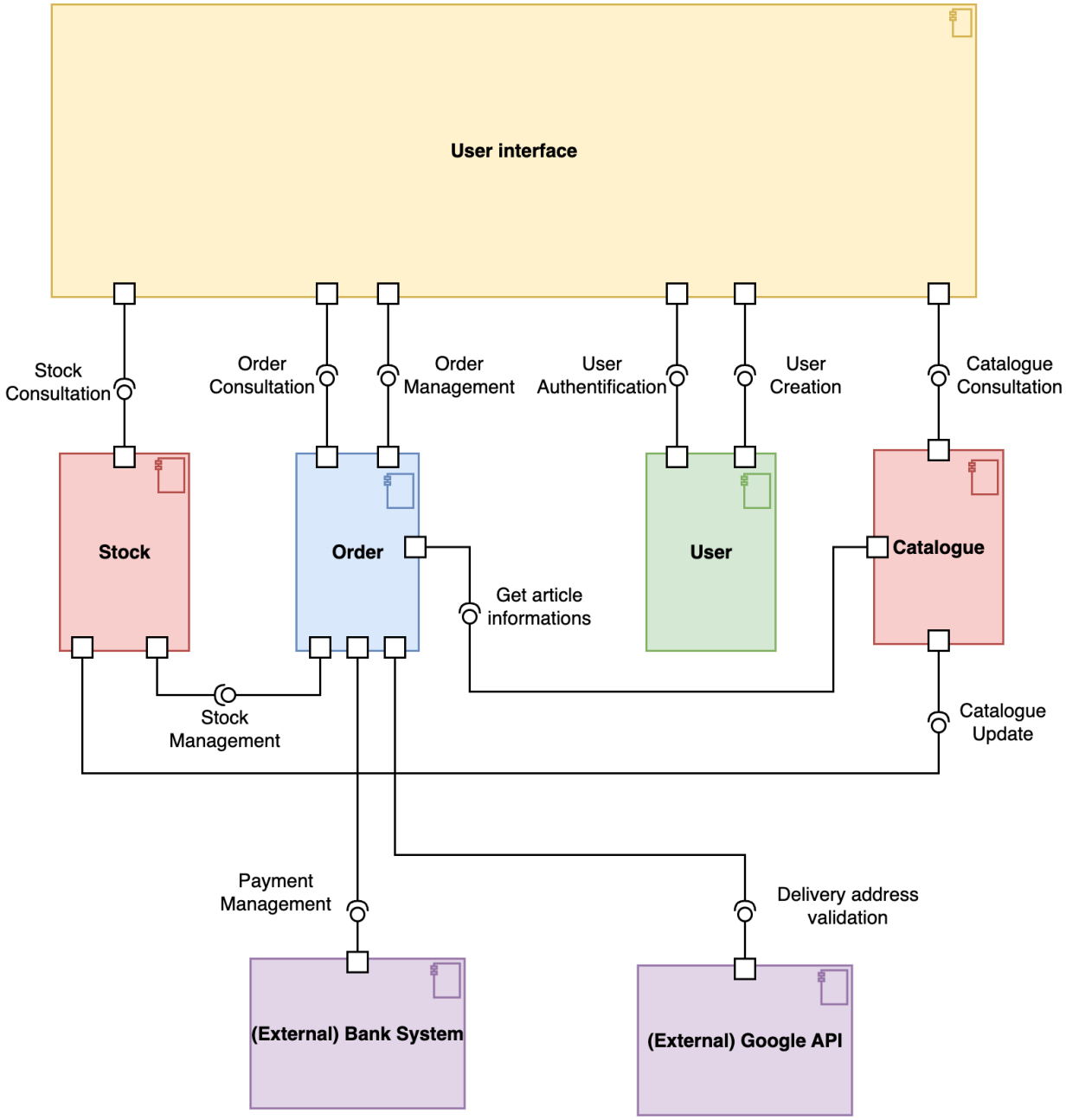
## 3.1. Composants généraux

### *3.1.1. Définition*

Les diagrammes de composants sont utilisés pour modéliser les composants qui contribuent à la réalisation de ses fonctionnalités. Les diagrammes de composants sont utilisés pour visualiser l'organisation des composants du système et les relations de dépendance entre eux.

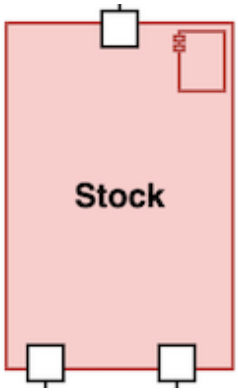
La page qui suit présente le diagramme de composants. Ensuite est indiquée la légende qui lui correspond.

### *3.1.2.Diagramme de composants*



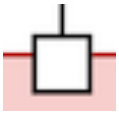
### *3.1.3.Légende*

- **Composant :**




Un composant est représenté par un rectangle comprenant deux autres rectangles sur le côté gauche.

- **Port :**



Un port est représenté par un carré blanc sur le bord d'un comptant, ce qui lui permet d'être relié à d'autres composants

- **Interface :**

L'interface se caractérise par un cercle connecté à un demi-cercle.

Le composant relié au demi-cercle est celui qui agit.

Le composant relié au cercle est celui qui est requis pour l'action.

### *3.1.4.Description des composants*

- **User interface :** Gère l'interface utilisateur. C'est le point d'entrée de l'application. Il comporte tous les comptes utilisateurs (clients, hôte de caisse, pizzaiolo, gérant etc.). Il est possible de se connecter à son compte utilisateur et créer un compte utilisateur. Il permet à un utilisateur (connecté ou non) d'accéder au catalogue des articles proposés par un restaurant. En fonction du user ID, l'utilisateur peut accéder à sa commande (en cours ou passée) ou à la gestion des commandes (en cours ou passée). Il permet également aux utilisateurs concernés de consulter les stocks d'un restaurant.

- **User :** Il s'agit du compte d'un utilisateur. La Graphic User Interface (GUI) changera en fonction du user ID rattaché au compte. Par exemple, si le user ID est un pizzaiolo, il y aura un certains nombre de boutons affichés à l'écran pour préparer la commande.

- **Catalogue :** Il liste l'ensemble des articles proposés par un restaurant. Ce composant agit sur le composant Stock. En effet, si un article n'est plus proposé à la carte, ou encore si un article est rajouté, le stock du restaurant devra s'adapter respecter son offre.

- **Order :** Il gère les commandes. Il inclut la constitution du panier d'achat, le statut d'une commande et la facturation. Ce composant agit sur le composant Catalogue pour accéder aux ID des articles. Il a git également sur le composant Google API pour valider l'adresse de livraison et sur BankSystem pour la gestion du paiement.

- **Stock :** Il contient les quantités ingrédients et des articles contenus dans le stock. Le stock est automatiquement mit à jour en fonction des commandes passées.

- **Bank System :** Il s'agit d'un composant externe en charge de la validation des règlement des commande.

- **Google API :** Il s'agit d'un composant externe qui gère la validation des adresses de livraison pour chaque commande.

# 4. Architecture de Déploiement

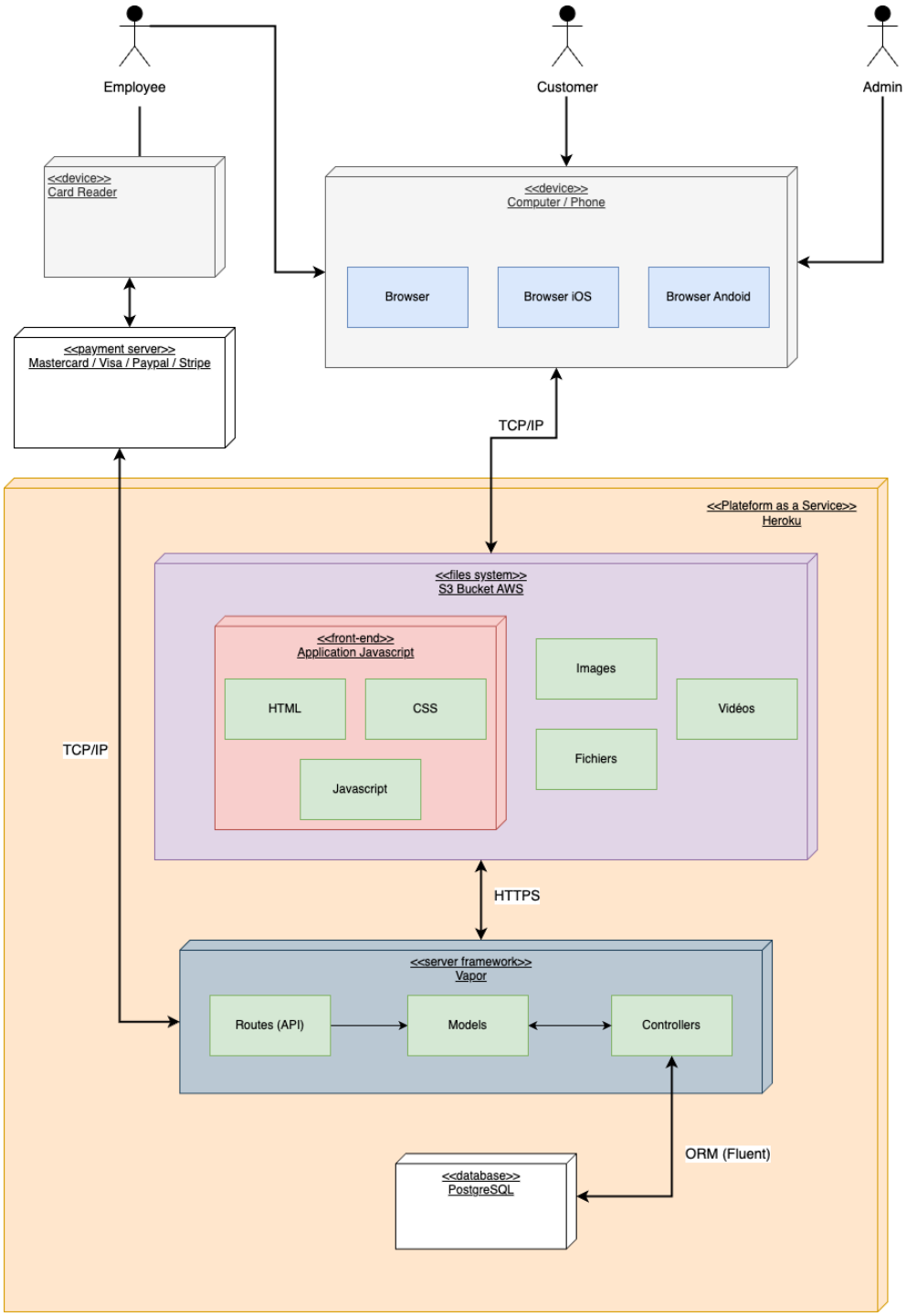
## 4.1. Déploiement

### *4.1.1. Définition*

Un diagramme de déploiement, également appelé diagramme de déploiement physique ou diagramme de déploiement de système, est un type de diagramme utilisé pour représenter la configuration matérielle et logicielle d'un système informatique ou d'un réseau. Il montre comment les différents composants d'un système sont connectés et comment ils interagissent entre eux. Les diagrammes de déploiement peuvent inclure des informations telles que les ordinateurs, les serveurs, les périphériques, les réseaux et les logiciels utilisés dans le système.

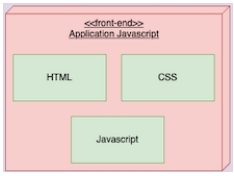
La page qui suit présente le diagramme de déploiement. Ensuite est indiquée la légende qui lui correspond.

### 4.1.2.Diagramme de déploiement



### *4.1.3.Légende*

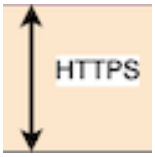
- **Noeud :**




Un nœud représente un élément physique ou logiciel qui fait partie du système représenté. Il peut s'agir d'un ordinateur, d'un serveur, d'un périphérique, d'un réseau ou d'un logiciel.

Les nœuds peuvent également être groupés en sous-ensembles pour montrer les relations hiérarchiques entre les différents éléments du système.

- **Connexions :**




Les nœuds sont reliés entre eux par des flèches pour indiquer les interactions entre les différents éléments. Par exemple, une flèche pourrait montrer comment un ordinateur accède à un serveur ou comment un périphérique est connecté à un réseau.

- **Fichiers :**


Les fichiers correspondent généralement aux différents composants logiciels ou matériels qui sont utilisés dans l'architecture de déploiement. Ces fichiers peuvent inclure des programmes d'application, des scripts de configuration, des fichiers de données, des images de machines virtuelles, des modèles de déploiement, etc.

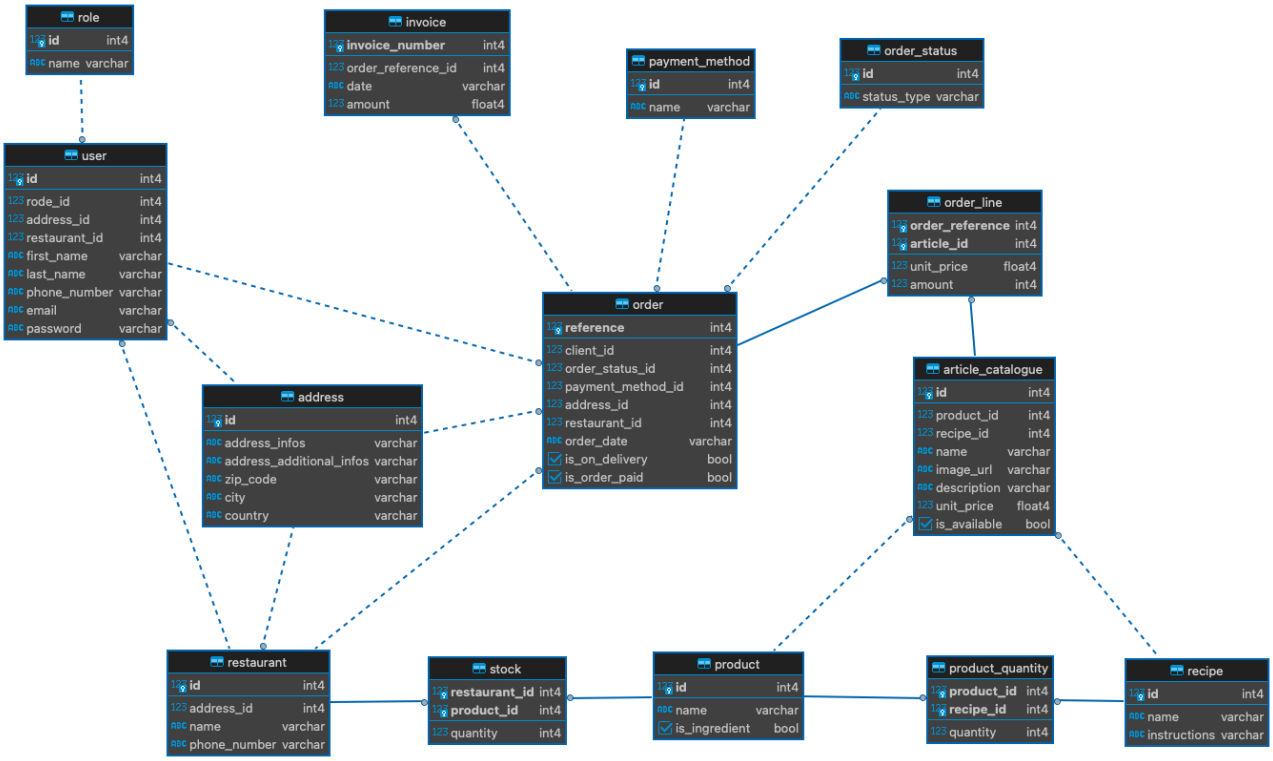
## 4.2.Serveur de base de données et de fichiers

### *4.2.1.PostgreSQL*

Le SGBDR désigné pour mettre en place ce système de gestion de commandes pour OC Pizza est PostgreSQL 15. Les interactions avec ce SGBDR seront gérer via l'ORM Fluent intégré au framework server Vapor.

La base de données sera hébergé sur Heroku. Pour cela, on utilisera le service de base de données relationnelle géré basé sur PostgreSQL.

### *4.2.2.Diagramme de la base de données*



### 4.2.3.Modélisation des données

La modélisation des différents objets et donc des tables se fera via DBeaver. Il peut être utilisé pour effectuer des tâches de gestion de base de données courantes, telles que la création, la modification et la suppression de tables, la saisie et la modification de données, la sauvegarde et la restauration de bases de données, l'exécution de requêtes SQL, et plus encore.

Il est aussi possible de gérer la base de données Postgres directement depuis l'interface Heroku ou via l'utilisation de l'outil de ligne de commande Heroku CLI.

### 4.2.4.S3 Bucket AWS

Amazon S3 (Simple Storage Service) est un service de stockage cloud géré par Amazon qui permet de stocker et de récupérer des objets, comme des fichiers, des images, des vidéos, etc. Les objets sont stockés dans des "buckets" (conteneurs) qui sont des répertoires logiques dans le système de stockage S3.

L'application peut accéder aux données stockées dans le bucket S3 via une API S3 fournie par Amazon. Les développeurs peuvent configurer les autorisations d'accès pour les données stockées dans le bucket S3 pour garantir la sécurité des données. Les données stockées dans le bucket S3 peuvent également être sauvegardées et restaurées en cas de besoin.

## 4.3.Serveur distant

### *4.3.1.Heroku*

Heroku est un service d'hébergement cloud qui permet aux développeurs de déployer, exécuter et gérer des applications web et des services en utilisant une variété de technologies de développement, comme Ruby, Java, Node.js, et Swift.

Les applications déployées sur Heroku sont hébergées sur des serveurs distants gérés par Heroku, qui sont physiquement situés dans des centres de données répartis dans le monde. Les utilisateurs peuvent accéder à ces applications via Internet, en utilisant un navigateur web ou en interagissant avec ces applications à l'aide d'une API. Les administrateurs système de Heroku sont responsables de la maintenance, de la mise à jour et de la sécurité des serveurs et des applications.

Heroku offre également la possibilité de connecter et de gérer des services externes tels que S3 bucket pour stocker les données de l'application.

### *4.3.2.Serveur Vapor*

Vapor est un framework web open-source pour le développement d'applications web en utilisant le langage de programmation Swift. Il a été conçu pour faciliter le développement d'applications web en utilisant les avantages de Swift tels que la sécurité de type statique, la performance et la facilité d'utilisation. Vapor offre un ensemble de bibliothèques et d'outils pour la gestion des requêtes HTTP, la gestion de la base de données, la gestion des sessions utilisateur, la validation des formulaires, la gestion des erreurs, et plus encore.

# 5. ARCHITECTURE LOGICIELLE

## 5.1. Principes généraux

Les sources et versions du projet sont gérées par **Git** et uploadés sur la plateforme collaborative **GitHub**. L'architecture logicielle répondra à la structure **MVC** en séparant la logique de la vue ainsi que des données.

### *5.1.1. Les modules*

L'architecture applicative respectera le MVC (Modèle-Vue-Contrôleur) et sera donc la suivante :

- **Modèle :** La couche modèle gère les données et la logique métier de l'application. Il contient les classes de modèles qui décrivent les données de l'application, les relations entre les données et les opérations de base de données courantes telles que l'insertion, la mise à jour et la suppression de données. Il peut également inclure des classes de services qui encapsulent les logiques métier spécifiques de l'application.

- **Vue :** La couche vue gère l'affichage des données de l'application. Il contient les templates, les composants visuels, les scripts et les styles qui définissent l'apparence de l'application. Il peut être implémenté en utilisant des frameworks de rendu côté client tels que React.

- **Contrôleur :** La couche contrôleur gère les interactions entre les utilisateurs et l'application. Il reçoit les requêtes HTTP, traitement les données, et retournent les réponses appropriées. Il peut utiliser les classes de modèles et de services pour accéder aux données et aux logiques métier de l'application.

### 5.1.2.Structure des sources

La structuration des répertoires du projet suit la logique suivante :

```
.
├── .gitignore
├── Package.swift
├── Procfile
├── README.md
├── Resources/
│   ├── Migrations/
│   └── Views/
├── Sources/
│   ├── App/
│   │   ├── Controllers/
│   │   ├── Models/
│   │   ├── Middleware/
│   │   ├── Public/
│   │   ├── Resources/
│   │   ├── Routes/
│   │   ├── Utils/
│   │   ├── configure.swift
│   │   └── main.swift
│   ├── Run/
│   └── Tests/
├── Public/
│   ├── styles/
│   ├── scripts/
│   └── images/
├── .heroku/
│   ├── Procfile
│   ├── runtime.txt
│   └── env
```

# 6.POINTS PARTICULIERS

## 6.1.Gestion des logs

Heroku fournit une plateforme de gestion de logs qui permet de collecter, de stocker et de visualiser les logs de l'application.

En utilisant Vapor, vous pouvez configurer vos contrôleurs et vos modèles pour écrire des logs à l'aide des fonctionnalités de journalisation de Vapor, c'est-à-dire en utilisant les fonctions **`print`**, **`dump`**, **`logger.debug()`**, **`logger.info()`**, **`logger.warning()`**, **`logger.error()`**, etc. Ces logs sont automatiquement collectés par Heroku et peuvent être visualisés à l'aide de la commande heroku logs ou en utilisant l'interface web de Heroku.

## 6.2.Fichiers de configuration

### *6.2.1.Configuration de l'application*

Les fichiers de configuration principaux de l'application serveur fonctionnant via le framework Vapor écrit en Swift sont:

- **`Package.swift`** : Ce fichier contient les informations de package pour votre application, telles que les dépendances et les cibles. Il est utilisé pour gérer les dépendances de l'application et pour configurer les cibles de compilation pour l'application.
- **`configure.swift`** : Ce fichier contient la configuration globale de l'application. Il est utilisé pour configurer les paramètres tels que les routes, les services, les middlewares, etc.
- **`routes.swift`** : Ce fichier contient les définitions de routes pour l'application. Il est utilisé pour configurer les chemins d'URL et les actions associées pour les requêtes HTTP.
- **`services.swift`** : Ce fichier contient les définitions de services pour l'application. Il est utilisé pour configurer les services tels que les bases de données, les services de stockage, les services de notification, etc.
- **`middlewares.swift`** : Ce fichier contient les définitions de middlewares pour l'application. Il est utilisé pour configurer les middlewares tels que l'authentification, la validation, la sécurité, etc.
- **`env.swift`** : Ce fichier contient les variables d'environnement pour l'application. Il est utilisé pour configurer les paramètres tels que les clés d'API, les URLs de base, les paramètres de base de données, etc.

### 6.2.2.Configuration de la base de données

Les fichiers d'importations de données sont fournis avec les différents script SQL sour le même format (.sql) au sein d'un fichier compressé (.zip) disponible sur le depository Github à l'adresse suivante :

*https://github.com/ocpizzasystem/dumps*

## 6.3.Ressources

Les ressources concernent tout les fichiers tels que les images et autres ressources annexes nécessaires au bon fonctionnement de l'application. Ces fichiers seront compressés au bon format au moment de l'upload. Cet upload sera fait via l'interface administrateur.

## 6.4.Environnement de développement

Le principal environnement de développement (IDE) utilisé est Xcode. Tout le code Swift de back-end et front-end sera implémenté via ce dernier. Git par le biais de Github est également interfacé par Xcode.

## 6.5.Procédure de packaging / livraison

### 6.5.1.Livraison

IT Consulting & Development assistera OC PIZZA tout au long de l'implémentation du système. Le package de livraison est un fichier ".zip" envoyé par "wetransfer.com". Il contiendra le projet Xcode ainsi que les fichiers de configurations de la base de données.

L'ensemble du projet sera également disponible sur Github (voir PV de livraison.

### 6.5.2.Formation

IT Consulting & Development assistera OC PIZZA dans l'a prise en main de son outil par à l'aide de formation en groupe pour les employés du restaurant, qui couvre les fonctionnalités clés de la solution, comme la gestion des commandes, la gestion des stocks et les rapports de ventes.

### *6.5.3.Suivi*

IT Consulting & Development organisera des réunions de suivi avec OC PIZZA pour évaluer l'efficacité de la formation et identifier les besoins de formation supplémentaires. Il est important de continuer à offrir un support après la formation pour assurer la réussite de l'utilisation de la solution.

### *6.5.4.Support*

IT Consulting & Development propose un support de la solution. L'objectif est de fournir un support technique pour aider OC PIZZA à résoudre les problèmes éventuels et à répondre à ses questions. Voici les service du super :

- **Mises à jour et correctifs** : Il est important de maintenir la solution en effectuant des mises à jour régulières et en résolvant les problèmes signalés par le client.
- **Assistance à distance** : Il est important de proposer une assistance à distance pour aider le client à résoudre les problèmes à distance, en utilisant des outils tels que les accès à distance ou les sessions de support en ligne.
- **Assistance en cas d'incident** : Il est important de proposer une assistance en cas d'incident pour aider le client à récupérer rapidement en cas de panne ou de perte de données.

Pour contacter le service du support, voici le lien et les éléments d'accès :

*https://itconsulting/helpdesk.com*

*Identifiant : OC_pizza_system_2023*
*Mot de passe : JTdfo-36529-te%op0*