Famery_Valentin_3_dossier_d’exploitation_0520

Date de création	@1 mai 2023 18:07
Étiquettes	

OC Pizza

Mise en place d’un nouveau système informatique pour l’ensemble des pizzerias du groupe.

Dossier d'exploitation

Version 1.0

Auteur

Famery Valentin

Lead Developer

Table des matières

1 - Versions

Auteur	Date	Description	Version
Famery Valentin	07/05/2023	Création du document	1.0

2 - Introduction

2.1 - Objet du document

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

Un document d'exploitation est un document qui vise à décrire de manière détaillée les opérations et les processus impliqués dans le fonctionnement d'un système, d'un équipement ou d'un logiciel. Il peut être utilisé comme guide de référence pour les utilisateurs, les administrateurs système ou les techniciens de maintenance. Le document d'exploitation contient généralement des informations sur les fonctionnalités du système, les procédures de configuration et d'installation, les exigences matérielles et logicielles, les problèmes courants et les solutions de dépannage, ainsi que des exemples pratiques d'utilisation. L'objectif principal d'un document d'exploitation est de fournir aux utilisateurs les informations nécessaires pour utiliser le système de manière efficace et sécurisée, tout en minimisant les temps d'arrêt et les erreurs.

2.2 - Références

Pour de plus amples informations, se référer :

1. **DCT -** Famery_Valentin_2_dossier_technique_052023 : Dossier de conception technique de l'application
2. **DCF -** Famery_Valentin_1_dossier_fonctionnelle_052023 : Dossier de conception fonctionnelle de l'application

3 - Pré-requis

3.1 - Système

3.1.1 - Serveur de Base de données

Serveur de base de données hébergeant le/les schémas/base Xxxx….

Google Cloud SQL est un service de base de données relationnelle entièrement géré qui permet aux utilisateurs de déployer, de gérer et d'évoluer des bases de données MySQL, PostgreSQL et SQL Server dans le cloud de Google.

Les utilisateurs peuvent choisir d'utiliser une base de données MySQL, PostgreSQL ou SQL Server pour stocker et gérer leurs données. Ils peuvent également choisir entre les éditions standard ou haute disponibilité (HA) de Cloud SQL en fonction de leurs besoins en termes de disponibilité et de performances.

L'édition standard est conçue pour les charges de travail de base, tandis que l'édition HA est conçue pour les charges de travail critiques nécessitant une disponibilité élevée. L'édition HA utilise la réplication synchrone pour fournir une disponibilité continue même en cas de panne de l'instance principale.

Google Cloud SQL est entièrement géré, ce qui signifie que Google s'occupe de toutes les tâches de maintenance, de mise à jour et de sécurité, telles que la surveillance, la sauvegarde, la mise à jour logicielle, la gestion des correctifs de sécurité, la restauration après sinistre, etc. Cela permet aux utilisateurs de se concentrer sur leur application et leurs données, sans avoir à se soucier de la gestion de l'infrastructure sous-jacente.

Cloud SQL est également hautement évolutif, ce qui signifie que les utilisateurs peuvent facilement augmenter ou diminuer les ressources de leur base de données en fonction des besoins de leur application. Cela permet de garantir des performances optimales à mesure que la charge de travail évolue.

En outre, Google Cloud SQL offre une intégration transparente avec d'autres services cloud de Google, tels que Google App Engine, Google Kubernetes Engine et Google Cloud Functions, permettant aux utilisateurs de créer des applications hautement performantes et évolutives avec une infrastructure de base de données solide.

Enfin, Google Cloud SQL offre également des fonctionnalités de sécurité avancées, telles que la gestion des connexions, l'authentification à deux facteurs, la protection des données en transit et au repos, et bien plus encore, pour garantir que les données des utilisateurs sont toujours protégées.

3.2 - Bases de données

3.2.1 - Scripts SQL création de la base de données :

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO

-- -----------------------------------------------------
```

```
-- Schema mydb
-- -----------------------------------------------------
DROP DATABASE IF EXISTS `mydb`;
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;



-- -----------------------------------------------------
-- Table `mydb`.`address`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`address` (
  `id` INT NOT NULL,
  `street` VARCHAR(200) NULL,
  `additional_address` VARCHAR(200) NULL,
  `postal_code` VARCHAR(200) NULL,
  `city` VARCHAR(200) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`user`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`user` (
  `id` INT NOT NULL,
  `last_name` VARCHAR(200) NULL,
  `first_name` VARCHAR(200) NULL,
  `date_create_account` DATE NULL,
  `phone_number` VARCHAR(200) NULL,
  `password` VARCHAR(200) NULL,
  `email` VARCHAR(200) NULL,
  `username` VARCHAR(250) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`client`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`client` (
  `id_user` INT NOT NULL,
  `id_address` INT NOT NULL,
  INDEX `fk_Client_Address1_idx` (`id_address` ASC) VISIBLE,
  INDEX `fk_Client_User1_idx` (`id_user` ASC) VISIBLE,
  PRIMARY KEY (`id_user`),
  CONSTRAINT `fk_Client_Address1`
    FOREIGN KEY (`id_address`)
    REFERENCES `mydb`.`address` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Client_User1`
    FOREIGN KEY (`id_user`)
    REFERENCES `mydb`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`point_of_sale`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`point_of_sale` (
  `id` INT NOT NULL,
  `name` VARCHAR(200) NULL,
  `siret` VARCHAR(200) NULL,
  `id_address` INT NOT NULL,
  `manager_id_user` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_PointOfSale_Address_idx` (`id_address` ASC) VISIBLE,
  INDEX `fk_PointOfSale_Employee1_idx` (`manager_id_user` ASC) VISIBLE,
  CONSTRAINT `fk_PointOfSale_Address`
    FOREIGN KEY (`id_address`)
    REFERENCES `mydb`.`address` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_PointOfSale_Employee1`
    FOREIGN KEY (`manager_id_user`)
    REFERENCES `mydb`.`employee` (`id_user`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`employee`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`employee` (
  `id_user` INT NOT NULL,
  `role` ENUM('Manager', 'Pizzaiolo', 'Delivery Man') NULL,
  `id_point_of_sale` INT NULL,
  PRIMARY KEY (`id_user`),
  INDEX `fk_Employee_User1_idx` (`id_user` ASC) VISIBLE,
  INDEX `fk_Employee_PointOfSale1_idx` (`id_point_of_sale` ASC) VISIBLE,
  CONSTRAINT `fk_Employee_User1`
    FOREIGN KEY (`id_user`)
    REFERENCES `mydb`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Employee_PointOfSale1`
    FOREIGN KEY (`id_point_of_sale`)
    REFERENCES `mydb`.`point_of_sale` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`order`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`order` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `date_create` DATETIME NOT NULL DEFAULT '2022-01-01 00:00:00',
  `date_delivery` DATETIME NOT NULL DEFAULT '2022-01-01 00:00:00',
  `status` ENUM('Created', 'In preparation', 'In the course of delivery', 'Delivered','Cancel') NOT NULL,
  `pay_method` ENUM('Credit Card', 'Cash') NULL,
  `amount_ht` FLOAT NULL,
  `amount_ttc` FLOAT NULL,
  `client_id_user` INT NOT NULL,
  `pizzaiolo_id_user` INT NOT NULL,
  `delivery_man_id_user` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Order_Client1_idx` (`client_id_user` ASC) VISIBLE,
  INDEX `fk_Order_Employee1_idx` (`pizzaiolo_id_user` ASC) VISIBLE,
  INDEX `fk_Order_Employee2_idx` (`delivery_man_id_user` ASC) VISIBLE,
  CONSTRAINT `fk_Order_Client1`
    FOREIGN KEY (`client_id_user`)
    REFERENCES `mydb`.`client` (`id_user`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Order_Employee1`
    FOREIGN KEY (`pizzaiolo_id_user`)
    REFERENCES `mydb`.`employee` (`id_user`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Order_Employee2`
    FOREIGN KEY (`delivery_man_id_user`)
    REFERENCES `mydb`.`employee` (`id_user`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`product`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`product` (
  `id` INT NOT NULL,
  `name` VARCHAR(200) NULL,
  `size` VARCHAR(200) NULL,
  `composition` VARCHAR(200) NULL,
  `category` VARCHAR(200) NULL,
  `unit_price_ht` FLOAT NULL,
  `unit_price_ttc` FLOAT NULL,
  `recipe` VARCHAR(20000) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`ingredient`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`ingredient` (
  `id` INT NOT NULL,
  `name` VARCHAR(200) NULL,
  PRIMARY KEY (`id`))
```

```
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`stock`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`stock` (
  `id` INT NOT NULL,
  `date_purchase` DATE NULL,
  `date_expiration` DATE NULL,
  `id_point_of_sale` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Stock_PointOfSale1_idx` (`id_point_of_sale` ASC) VISIBLE,
  CONSTRAINT `fk_Stock_PointOfSale1`
    FOREIGN KEY (`id_point_of_sale`)
    REFERENCES `mydb`.`point_of_sale` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`assoc_order_product`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`assoc_order_product` (
  `id_order` INT NOT NULL,
  `id_product` INT NOT NULL,
  `quantity` INT NULL,
  INDEX `fk_assoc_Order_Product_Order1_idx` (`id_order` ASC) VISIBLE,
  INDEX `fk_assoc_Order_Product_Product1_idx` (`id_product` ASC) VISIBLE,
  PRIMARY KEY (`id_order`, `id_product`),
  CONSTRAINT `fk_assoc_Order_Product_Order1`
    FOREIGN KEY (`id_order`)
    REFERENCES `mydb`.`order` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_assoc_Order_Product_Product1`
    FOREIGN KEY (`id_product`)
    REFERENCES `mydb`.`product` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`assoc_ingredient_stock`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `mydb`.`assoc_ingredient_stock` (
  `id_stock` INT NOT NULL,
  `id_ingredient` INT NOT NULL,
  `quantity` INT NULL,
  `unit` VARCHAR(200) NULL,
  INDEX `fk_assoc_Ingredient_Stock_Stock1_idx` (`id_stock` ASC) VISIBLE,
  INDEX `fk_assoc_Ingredient_Stock_Ingredient1_idx` (`id_ingredient` ASC) VISIBLE,
  PRIMARY KEY (`id_stock`, `id_ingredient`),
  CONSTRAINT `fk_assoc_Ingredient_Stock_Stock1`
    FOREIGN KEY (`id_stock`)
    REFERENCES `mydb`.`stock` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_assoc_Ingredient_Stock_Ingredient1`
    FOREIGN KEY (`id_ingredient`)
    REFERENCES `mydb`.`ingredient` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `mydb`.`assoc_product_ingredient` (
  `id_product` INT NOT NULL,
  `id_ingredient` INT NOT NULL,
  `quantity_required` INT NULL,
  `unit` VARCHAR(43) NULL,
  INDEX `fk_assoc_product_ingredient_product_idx` (`id_product` ASC) VISIBLE,
  INDEX `fk_assoc_product_ingredient_ingredient_idx` (`id_ingredient` ASC) VISIBLE,
  PRIMARY KEY (`id_product`, `id_ingredient`),
  CONSTRAINT `fk_assoc_product_ingredient_product`
    FOREIGN KEY (`id_product`)
    REFERENCES `mydb`.`Product` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_assoc_product_ingredient_ingredient`
    FOREIGN KEY (`id_ingredient`)
    REFERENCES `mydb`.`Ingredient` (`id`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

3.2.2 - Scripts SQL insertion dans la base de données :

```
INSERT INTO `mydb`.`address` (id, street, additional_address, postal_code, city) VALUES
(1, 'Rue du Faubourg Saint-Honoré', NULL, '75008', 'Paris'),
(2, 'Avenue des Champs-Élysées', NULL, '75008', 'Paris'),
(3, 'Rue de la Paix', NULL, '75002', 'Paris');

INSERT INTO `mydb`.`user` (`id`, `last_name`, `first_name`, `date_create_account`, `phone_number`, `password`, `email`, `username`) VA
(1, 'Smith', 'John', '2022-01-01', '1234567890', 'mypassword', 'john.smith@example.com', 'jsmith'),
(2, 'Doe', 'Jane', '2022-01-02', '2345678901', 'mypassword', 'jane.doe@example.com', 'jdoe'),
(3, 'Johnson', 'Michael', '2022-01-03', '3456789012', 'mypassword', 'michael.johnson@example.com', 'mjohnson'),
(4, 'Lee', 'Sarah', '2022-01-04', '4567890123', 'mypassword', 'sarah.lee@example.com', 'slee'),
(5, 'Brown', 'David', '2022-01-05', '5678901234', 'mypassword', 'david.brown@example.com', 'dbrown');

INSERT INTO `mydb`.`client` (id_user, id_address) VALUES
(1, 1);

INSERT INTO `mydb`.`employee` (`id_user`, `role`, `id_point_of_sale`) VALUES
(2, 'Manager', NULL),
(3, 'Pizzaiolo', NULL),
(4, 'Delivery Man', NULL);

INSERT INTO `mydb`.`point_of_sale` (`id`, `name`, `siret`, `id_address`, `manager_id_user`) VALUES
(1, 'Pizza Place 1', '123456789', 2, 2);


update `mydb`.`employee` set `id_point_of_sale` = 1 where `id_user` = 2;
update `mydb`.`employee` set `id_point_of_sale` = 1 where `id_user` = 3;
update `mydb`.`employee` set `id_point_of_sale` = 1 where `id_user` = 4;

INSERT INTO `mydb`.`order` (`date_create`, `date_delivery`, `status`, `pay_method`, `amount_ht`, `amount_ttc`, `client_id_user`, `pizz
VALUES ('2022-03-31 15:30:00', '2022-03-31 18:45:00', 'In preparation', 'Credit Card', 25.0, 30.0, 1, 3, 4);

INSERT INTO mydb.product (id, name, size, composition, category, unit_price_ht, unit_price_ttc, recipe) VALUES
(1, 'Pizza Hawaïenne', 'Large', 'Jambon, Ananas, Fromage', 'Pizza salée', 10.00, 12.00, "1. Préchauffez le four à 220°C (thermostat 7-
(2, 'Pizza Chocolat-banane', 'Medium', 'Chocolat, Banane, Sucre', 'Pizza sucrée', 8.00, 10.00, '1. Préchauffez le four à 200°C (thermo

INSERT INTO `mydb`.`ingredient` (id, name) VALUES
(1, 'Jambon'),
(2, 'Ananas'),
(3, 'Fromage'),
(4, 'Chocolat'),
(5, 'Banane'),
(6, 'Sucre'),
(7, 'Saucisse'),
(8, 'Champignons'),
(9, 'Poulet'),
(10, 'Poivrons'),
(11, 'Oeuf'),
(12, 'Noix de coco'),
(13, 'Miel');

INSERT INTO `mydb`.`assoc_product_ingredient` (id_product,id_ingredient,quantity_required,unit)VALUES
(1,1,50,'g'),
(2,1,40,'g'),
(3,1,60,'g'),
(4,2,56,'g'),
(5,2,72,'g'),
(6,2,90,'g'),
(7,1,30,'g');

INSERT INTO mydb.assoc_order_product (id_order, id_product, quantity) VALUES (1, 1, 2),(1,2,3);

INSERT INTO `mydb`.`stock` (`id`, `date_purchase`, `date_expiration`, `id_point_of_sale`)
VALUES (1, '2022-02-01', '2022-05-01', 1),
       (2, '2022-03-15', '2022-06-15', 1);

INSERT INTO `mydb`.`assoc_ingredient_stock` (`id_stock`, `id_ingredient`, `quantity`, `unit`)
VALUES (1, 1, 500, 'g'),
       (2, 2, 100, 'g');
```

3.3 - Web-services

- Stripe : https://stripe.com pour les paiements
- DartFrog : https://dartfrog.vgv.dev le coeur de l’application backend
- MySQL/Google Cloud SQL : https://cloud.google.com/sql/docs/mysql?hl=fr le stockage de toutes les données de l’application

4 - Procédure de déploiement

4.1 - Déploiement de la base de données

Les étapes pour déployer une base de données MySQL sur Google Cloud SQL :

1. Connectez-vous à la console Google Cloud et créez un projet si vous n'en avez pas déjà un.
2. Dans le menu principal de la console, sélectionnez "SQL" sous la section "Stockage".
3. Cliquez sur le bouton "Créer une instance".
4. Choisissez "MySQL" comme type de base de données.
5. Choisissez le niveau de performance et la configuration de l'instance.
6. Choisissez une région pour votre instance.
7. Configurez les paramètres de connexion, y compris les noms d'utilisateur et les mots de passe pour les utilisateurs MySQL.
8. Cliquez sur "Créer" pour lancer le déploiement de votre instance.
9. Attendez que votre instance soit créée. Cela peut prendre quelques minutes.
10. Connectez-vous à votre instance en utilisant un client MySQL, comme MySQL Workbench ou la ligne de commande MySQL.
11. Créez votre base de données et vos tables.
12. Importez vos données dans la base de données en utilisant la commande "mysqlimport" ou en utilisant un client MySQL avec une fonctionnalité d'importation de données.

4.2 - Déploiement du backend

les étapes pour déployer une application Dart sur Google Cloud Run :

1. Tout d'abord, assurez-vous que votre application Dart est prête à être déployée. Vous pouvez créer une application Dart à l'aide du framework DartFrog ou en utilisant Dart seul.
2. Ouvrez la console Google Cloud et créez un nouveau projet si vous n'en avez pas déjà un.
3. Dans le menu principal de la console, sélectionnez "Cloud Run" sous la section "Calcul".
4. Cliquez sur "Créer un service" pour créer un nouveau service.
5. Donnez un nom à votre service et choisissez la région dans laquelle vous souhaitez déployer votre application.
6. Choisissez "Continuer sans sélectionner" pour utiliser le déploiement recommandé.
7. Sélectionnez "Publier un nouveau conteneur" et choisissez "Dart" comme langage.
8. Sélectionnez la méthode de déploiement que vous souhaitez utiliser, par exemple, "Déployer depuis une image".
9. Configurez les paramètres de votre conteneur, y compris le nom et la version de votre image de conteneur.
10. Cliquez sur "Créer" pour lancer le déploiement de votre application.
11. Attendez que votre application soit déployée. Cela peut prendre quelques minutes.
12. Testez votre application en utilisant l'URL fournie par Google Cloud Run.

4.3 - Déploiment du Frontend

4.3.1 - Web

Les étapes pour déployer un frontend Flutter Web sur Google Cloud Hosting :

1. Tout d'abord, assurez-vous que votre frontend Flutter Web est prêt à être déployé. Vous pouvez créer un frontend Flutter Web à l'aide de Flutter en utilisant la commande "flutter create ." dans votre répertoire de projet.
2. Ouvrez la console Google Cloud et créez un nouveau projet si vous n'en avez pas déjà un.
3. Dans le menu principal de la console, sélectionnez "Cloud Hosting" sous la section "Web".
4. Cliquez sur "Créer un site web" pour créer un nouveau site web.
5. Donnez un nom à votre site web et choisissez la région dans laquelle vous souhaitez déployer votre site.
6. Sélectionnez "Continuer sans configuration" pour utiliser la configuration recommandée.
7. Téléchargez le fichier de configuration de Firebase Hosting en cliquant sur le bouton "Télécharger".
8. Ouvrez le fichier "firebase.json" et modifiez le champ "public" pour qu'il pointe vers le dossier "build/web" de votre projet Flutter Web.
9. Enregistrez les modifications apportées au fichier "firebase.json".
10. Ouvrez une invite de commandes et naviguez jusqu'au répertoire de votre projet Flutter Web.
11. Exécutez la commande "flutter build web" pour générer le code de production de votre frontend Flutter Web.
12. Exécutez la commande "firebase deploy" pour déployer votre frontend Flutter Web sur Google Cloud Hosting.
13. Attendez que votre site web soit déployé. Cela peut prendre quelques minutes.
14. Testez votre site web en utilisant l'URL fournie par Google Cloud Hosting.

4.3.2 - Google Play Console

1. Créez un compte développeur sur Google Play Console si vous n'en avez pas déjà un.
2. Accédez à la Google Play Console avec votre compte développeur.
3. Cliquez sur "Créer une application" et suivez les étapes pour ajouter les informations de base de votre application, telles que le nom, la description et les captures d'écran.
4. Générez un fichier APK signé pour votre application à l'aide d'Android Studio ou de la ligne de commande.
5. Uploadez votre fichier APK signé dans Google Play Console.
6. Configurez les détails de la version de votre application, y compris le numéro de version et la cible de distribution.
7. Vérifiez que tous les paramètres de votre application sont corrects, tels que les autorisations, les prix et les langues prises en charge.
8. Soumettez votre application pour examen en cliquant sur le bouton "Soumettre à l'examen" dans Google Play Console.
9. Une fois que votre application est approuvée, vous pouvez la publier sur le Google Play Store.

4.3.2 - Apple Developper / Appstore

1. Inscrivez-vous à un compte Apple Developer Program si vous n'en avez pas déjà un.
2. Créez un certificat de distribution pour votre application à l'aide de l'outil "Keychain Access" sur votre Mac.
3. Générez un profil de distribution pour votre application via le site web du Apple Developer Program.
4. Ouvrez Xcode (l'environnement de développement intégré d'Apple) et configurez votre projet avec le profil de distribution.
5. Assurez-vous que votre application est prête à être distribuée en vérifiant les informations, les images d'aperçu et les métadonnées.
6. Archivez votre application dans Xcode pour créer un fichier d'archive.
7. Connectez-vous à votre compte Apple Developer Program sur le site web et accédez à l'App Store Connect.
8. Créez une nouvelle version de votre application et suivez les étapes pour ajouter les informations requises.

9. Soumettez votre fichier d'archive via App Store Connect pour examen.
10. Une fois que votre application est approuvée, vous pouvez la rendre disponible sur l'App Store.

5 - Procédure de démarrage / arrêt

5.1 - Base de données

5.1.1 - Démarrage d'une instance

1. Accédez à la console Google Cloud et sélectionnez votre projet.
2. Accédez à la page "Instances de base de données" dans la section "Base de données" du menu.
3. Sélectionnez l'instance que vous souhaitez démarrer en cochant la case à côté de son nom.
4. Cliquez sur le bouton "Démarrer" en haut de la page.
5. Sélectionnez le type de configuration de démarrage que vous souhaitez utiliser, puis cliquez sur "Démarrer".
6. Attendez que l'instance soit en ligne. Cela peut prendre plusieurs minutes.

5.1.2 - Arrêt d'une instance

1. Accédez à la page "Instances de base de données" dans la section "Base de données" du menu.
2. Sélectionnez l'instance que vous souhaitez arrêter en cochant la case à côté de son nom.
3. Cliquez sur le bouton "Arrêter" en haut de la page.
4. Sélectionnez le type de configuration d'arrêt que vous souhaitez utiliser, puis cliquez sur "Arrêter".
5. Attendez que l'instance soit arrêtée. Cela peut prendre plusieurs minutes.

5.1.3 - Avantages

1. Réduction des coûts : en arrêtant votre instance Cloud SQL lorsque vous n'en avez pas besoin, vous pouvez économiser des coûts d'infrastructure.
2. Sécurité : En arrêtant votre instance, vous réduisez les risques de vulnérabilité en exposant moins votre base de données sur internet.
3. Ressources efficaces : En arrêtant une instance Cloud SQL, vous libérez les ressources allouées pour l'instance, ce qui peut être utilisé pour d'autres charges de travail.

5.1.4 - Inconvénients

1. Temps de redémarrage : Si vous devez redémarrer votre instance Cloud SQL, cela peut prendre plusieurs minutes, ce qui peut causer des temps d'arrêt pour vos applications.
2. Perte de données : Si vous arrêtez votre instance sans sauvegarder les données, vous risquez de perdre toutes les données non sauvegardées.
3. Arrêt des opérations en cours : Si des opérations de base de données sont en cours, telles que des transactions ou des mises à jour, l'arrêt de l'instance peut interrompre ces opérations et entraîner une perte de données.

5.1.5 - Arrêt temporaire

Si vous arrêtez temporairement une instance Cloud SQL, cela peut être utile si vous n'avez pas besoin d'utiliser votre base de données pendant une courte période, par exemple pendant la nuit. Cela peut aider à réduire les coûts d'infrastructure.

Cependant, si vous arrêtez temporairement votre instance, vous devez prendre en compte les implications suivantes :

1. Il n'est pas recommandé d'arrêter une instance Cloud SQL pendant de longues périodes, car cela peut affecter les performances de votre base de données et augmenter les temps de redémarrage.
2. Assurez-vous de sauvegarder toutes les données avant d'arrêter votre instance, au cas où il y aurait des données non sauvegardées.

3. Si vous arrêtez votre instance, assurez-vous que les applications qui en dépendent sont conçues pour gérer l'indisponibilité temporaire de la base de données.

6 - Procedure de mis a jour / sauvegarde

La procédure de mise à jour d'une instance Cloud SQL sur Google Cloud dépend de la méthode d'installation que vous avez utilisée pour votre instance Cloud SQL. Voici les étapes générales pour effectuer une mise à jour :

1. Vérifiez la version actuelle de votre instance Cloud SQL. Pour cela, accédez à la page "Instances de base de données" dans la section "Base de données" du menu, sélectionnez l'instance que vous souhaitez mettre à jour et notez la version actuelle de la base de données.
2. Vérifiez les versions disponibles pour votre base de données. Les versions disponibles peuvent varier en fonction du type de base de données que vous utilisez (MySQL, PostgreSQL, etc.). Vous pouvez consulter les versions disponibles dans la documentation de Google Cloud.
3. Créez une nouvelle instance avec la version de la base de données que vous souhaitez utiliser. Pour cela, accédez à la page "Créer une instance de base de données" dans la section "Base de données" du menu, et sélectionnez la version de la base de données que vous souhaitez utiliser. Suivez les étapes pour configurer les paramètres de votre nouvelle instance.
4. Effectuez une sauvegarde de votre base de données actuelle. Pour cela, accédez à la page "Sauvegardes" dans la section "Base de données" du menu, sélectionnez votre instance Cloud SQL, puis cliquez sur le bouton "Créer une sauvegarde".
5. Exportez les données de votre instance Cloud SQL actuelle vers votre nouvelle instance. Pour cela, vous pouvez utiliser l'outil de migration de base de données fourni par Google Cloud ou une autre solution tierce.
6. Vérifiez que toutes les données ont été correctement exportées vers votre nouvelle instance.
7. Testez votre application pour vous assurer que tout fonctionne correctement avec la nouvelle version de la base de données.
8. Une fois que vous êtes sûr que tout fonctionne correctement avec la nouvelle instance, vous pouvez supprimer l'ancienne instance de votre base de données.

7 - Supervision des performances

7.1 - Base de données / Backend

Google Cloud Operations : c'est la solution de monitoring et de logging de Google Cloud. Elle permet de suivre les performances de vos applications Cloud Run et Cloud SQL en temps réel, de recevoir des alertes en cas de problèmes et de consulter des métriques sur la disponibilité, les performances et les erreurs.

Pour accéder à Google Cloud Operations, suivez ces étapes :

1. Connectez-vous à votre compte Google Cloud Console en utilisant votre adresse e-mail et votre mot de passe.
2. Une fois connecté, cliquez sur le menu de navigation de la console Cloud (l'icône hamburger en haut à gauche) et sélectionnez "Operations".
3. Si c'est votre première fois en accédant à Cloud Operations, vous pouvez suivre les instructions pour configurer votre compte. Si vous avez déjà configuré votre compte, vous devriez voir un tableau de bord avec des métriques et des graphiques sur la performance de vos ressources Google Cloud.
4. Dans le tableau de bord, vous pouvez visualiser des informations sur la disponibilité, les performances et les erreurs de vos ressources Google Cloud. Vous pouvez également créer des alertes pour surveiller les métriques importantes et recevoir des notifications par e-mail ou via des canaux de communication tiers tels que Slack.
5. Pour accéder à des fonctionnalités plus avancées, vous pouvez explorer les différentes sections de Cloud Operations, telles que les journaux (logs), les traces, les profils et les diagnostics. Chaque section fournit des informations sur les performances de vos ressources Google Cloud et vous permet de les analyser en profondeur.

8 - Gestion de versions

8.1 - Git

Initialisation du dépôt local : La première étape consiste à initialiser un dépôt Git local dans le répertoire de votre projet. Cela crée un espace où Git peut enregistrer les modifications de votre code.

```
$ git init
```

Ajout de fichiers : Après avoir initialisé le dépôt, vous devez ajouter les fichiers que vous souhaitez suivre et versionner avec Git. Vous pouvez ajouter des fichiers individuellement ou tous les fichiers d'un répertoire.

```
$ git add <nom_fichier>
```

Vous pouvez également utiliser la commande `git add .` pour ajouter tous les fichiers modifiés dans le répertoire actuel.

Validation des modifications : Une fois que vous avez ajouté les fichiers, vous devez valider les modifications en créant un commit. Un commit est une capture instantanée de l'état de votre code à un moment précis, accompagnée d'un message décrivant les modifications apportées.

```
$ git commit -m "Description des modifications”
```

Branches : Les branches permettent de travailler sur des versions distinctes du code. Par défaut, Git crée une branche principale appelée "master" (ou "main" depuis 2020) qui représente la version stable de votre code. Vous pouvez créer de nouvelles branches pour développer des fonctionnalités ou résoudre des problèmes, puis les fusionner avec la branche principale lorsque vous êtes prêt.

```
$ git branch <nom_branche>
```

```
$ git checkout <nom_branche>
```

Fusion de branches : Lorsque vous avez terminé de travailler sur une branche et que vous souhaitez intégrer les modifications dans la branche principale, vous pouvez effectuer une fusion. Cela combine les modifications de la branche actuelle avec la branche cible.

```
$ git checkout <branche_cible>
$ git merge <branche_source>
```

Récupération et envoi de modifications : Pour collaborer avec d'autres développeurs, vous pouvez récupérer les modifications qu'ils ont apportées à un dépôt distant. Vous pouvez également envoyer vos propres modifications vers un dépôt distant, généralement hébergé sur GitHub.

```
$ git pull origin <branche>
```

```
$ git push origin <branche>
```

Résolution de conflits : Lorsque Git ne parvient pas à fusionner automatiquement les modifications en raison de conflits, vous devez les résoudre manuellement. Vous devrez identifier les parties en conflit dans les fichiers, choisir les modifications appropriées et valider la résolution.

Suivi de l'historique et des modifications : Git conserve un historique complet de toutes les modifications apportées au code. Vous pouvez afficher l'historique des commits, comparer les versions, revenir à des versions antérieures et même effectuer des recherches spécifiques dans l'historique.

```
$ git log
```

```
$ git diff <commit1> <commit2>
```

8.2 - Github

Intégration avec GitHub : GitHub est une plateforme d'hébergement de dépôts Git qui facilite la collaboration et le partage de code. Vous pouvez créer un compte sur GitHub et créer un dépôt distant pour votre projet. Une fois le dépôt créé, vous pouvez le lier à votre dépôt local.

```
$ git remote add origin <URL_dépôt_GitHub>
```

Vous pouvez ensuite pousser vos modifications vers le dépôt distant sur GitHub.

```
$ git push origin <branche>
```

Pull requests : Lorsque vous souhaitez contribuer à un dépôt sur GitHub, vous pouvez proposer des modifications via une pull request. Une pull request est une demande d'intégration de vos modifications dans la branche principale du dépôt. Les autres collaborateurs peuvent examiner vos modifications, laisser des commentaires et décider de les fusionner.

Clonage d'un dépôt : Si vous souhaitez récupérer une copie complète d'un dépôt distant sur votre machine locale, vous pouvez le cloner à l'aide de la commande `git clone`. Cela crée une copie locale du dépôt, y compris toutes les branches et l'historique des commits.

```
$ git clone <URL_dépôt_GitHub>
```

Gestion des conflits sur GitHub : Lorsque des conflits surviennent lors d'une pull request, GitHub fournit une interface conviviale pour les résoudre directement sur la plateforme. Vous pouvez afficher les différences entre les branches, sélectionner les modifications à conserver et valider la résolution des conflits.

Tags : Les tags vous permettent de marquer des versions spécifiques de votre code pour des références ultérieures. Vous pouvez créer un tag pour une version stable, une version de publication ou toute autre étape importante de votre projet.

```
$ git tag <nom_tag>
```

```
$ git push --tags
```

9 - Glossaire

Stripe	Stripe est une plateforme de paiement en ligne qui permet aux entreprises de collecter des paiements sur internet. Elle

	offre des outils pour créer des pages de paiement, gérer les abonnements, gérer les remboursements, suivre les transactions et bien plus encore. Les clients peuvent payer avec leur carte de crédit, leur compte bancaire ou d'autres moyens de paiement.
Web-services	Un web service est un système informatique qui permet à des applications de communiquer et d'échanger des données via des protocoles web standardisés. Concrètement, un web service permet à une application d'exposer des fonctionnalités ou des données à d'autres applications à travers une interface web.
SQL	SQL (Structured Query Language) est un langage informatique standardisé utilisé pour gérer et manipuler des bases de données relationnelles. Il a été développé dans les années 1970 et est devenu le langage de choix pour les applications de gestion de données, en raison de sa simplicité, de sa flexibilité et de sa compatibilité avec la plupart des systèmes de gestion de bases de données relationnelles.
Serveur de base de données	Un serveur de base de données est un logiciel qui permet de stocker, d'organiser et de gérer les données d'une ou plusieurs applications. Il peut être installé sur une machine physique ou virtuelle, et est généralement accessible à distance via un réseau, afin que les utilisateurs puissent accéder aux données stockées dans la base de données.