**Rapport de Projet : Développement de l'Application Mobile pour Caluire Mobile**

**Table des matière**

1. Introduction

1.Listing des compétences abordé lors du projet

2. Remerciements

3.Résumer projet

1. Présentation de la Société Caluire Mobile
2. Environnement de travail

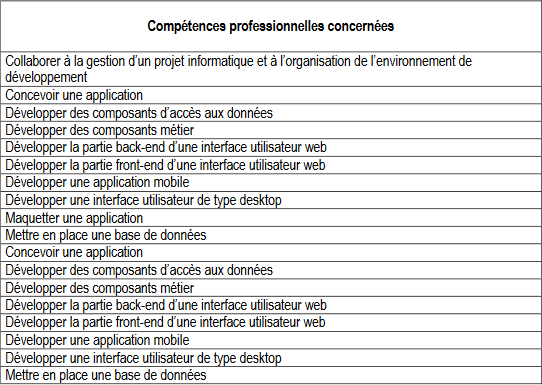
1. Outils utilisés

2. Technologie utilisés

1. Analyse et Conception
   1. Définition des Besoins
   2. Diagramme de Cas d'Utilisation
   3. Description des Cas d'Utilisation
   4. Diagramme de Séquence
   5. Diagramme de Classe d’Analyse
2. Analyse et Conception des Bases de Données
   1. Base de Données Locale (SQLite)
   2. Base de Données Distante (MySQL)
3. Phase de Développement
   1. Prise en Main de React Native CLI
   2. Mise en Place des Patterns
   3. Développement des Fonctionnalités
4. Utilisation de Mailjet pour l'Envoi de Mails
   1. Configuration de Mailjet
   2. Envoi de Mails avec Mailjet
   3. Vérification du Compte Client
5. Synchronisation et Sécurité
   1. Synchronisation des Données
   2. Sécurité des Données
6. Annexes

**1**. **Introduction**

1. Listing des compétences abordé lors du projet



**2.** Remerciements

3.Résumer projet

Ce projet vise à développer une application mobile dédiée à la vente et à la réparation de produits en ligne pour Caluire Mobile, une boutique de téléphonie. L'objectif est de fournir une expérience utilisateur fluide permettant de naviguer, acheter des produits ou programmer des réparations et des pickups depuis des appareils mobiles. Compatible avec iOS et Android, l'application sera construite en utilisant React Native CLI et MySQL pour la gestion des données, avec des API REST pour la communication entre le front-end (en React Native CLI) et le back-end (en .NET 8).

L'application inclura plusieurs fonctionnalités clés pour améliorer l'expérience client, telles que la création de comptes personnels, un système de paiement sécurisé et des options avancées de recherche et de navigation. Les utilisateurs pourront interagir via un système de chat pour communiquer directement avec les employés et bénéficier d'une section dédiée aux réparations et aux pickups, avec une option express pour un service plus rapide moyennant un supplément.

D'un point de vue technique, l'application devra offrir des performances élevées pour supporter un grand nombre d'utilisateurs simultanés, garantir la sécurité des données utilisateurs et des transactions, et utiliser du clean code, ce qui nous permettra également de rajouter des tests unitaires.

Les livrables du projet incluront l'application mobile fonctionnelle pour les plateformes spécifiées, ainsi qu'une documentation détaillée contenant le manuel d'utilisation et les instructions de déploiement. De nombreuses réunions ont eu lieu avec le client pour déterminer les besoins fonctionnels de l’application.

**2. Présentation de la Société Caluire Mobile**

Caluire Mobile est une jeune entreprise (crée en janvier2024) domicilié àcaluire et cuire dans les monts du lyonnais spécialisée dans la vente et la réparation de produits électroniques. L'entreprise vise à améliorer l'expérience client grâce à une application mobile moderne qui facilitera l'achat de produits et la gestion des services de réparation.

**3. Analyse et Conception**

**3.1. Définition des Besoins**

A la suite de plusieurs rendez-vous avec le client nous avons pu crée le cahier des charge ensemble(voir annexe) l'application doit permettre aux utilisateurs de :

* Naviguer et rechercher des produits facilement.
* Ajouter des produits à leur panier.
* Programmer des rendez-vous de réparation.
* Programmer des rendez-vous de pickUp.
* Communiquer avec les employer en temps réel.
* Effectuer des paiements sécurisés.

**3.2. Diagramme de Cas d'Utilisation**

Les principaux cas d'utilisation de l'application Caluire Mobile sont :

* Gestion des clients.
* Gestion des employés.
* Gestion des produits.
* Gestion des opérations.
* Gestion des rendez-vous.
* Gestion des transactions de paiement.
* Gestion des types de produits.
* Fonctionnalité de chat en temps réel.

**3.3. Description des Cas d'Utilisation**

Chaque cas d'utilisation décrit les interactions possibles entre les utilisateurs et le système pour accomplir une tâche spécifique, comme la synchronisation de la base de données, la gestion des produits, ou la programmation de rendez-vous.

**3.4. Diagramme de Séquence**

Le diagramme de séquence illustre les interactions temporelles entre les différents objets pour réaliser un cas d'utilisation particulier.

**3.5. Diagramme de Classe d’Analyse**

Le diagramme de classe montre les classes du système, leurs attributs, méthodes et les relations entre elles.

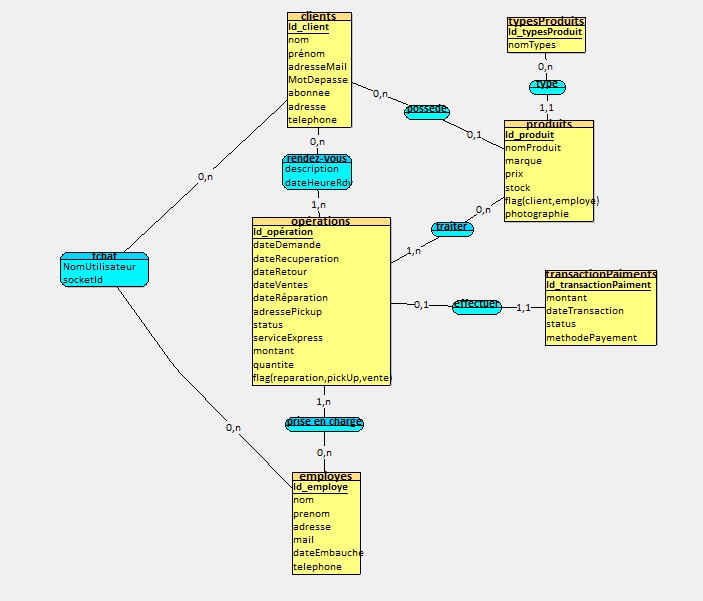
**4. Analyse et Conception des Bases de Données**

**4.1. Base de Données Locale (SQL)**

* **Modèle Conceptuel de Données (MCD) :**

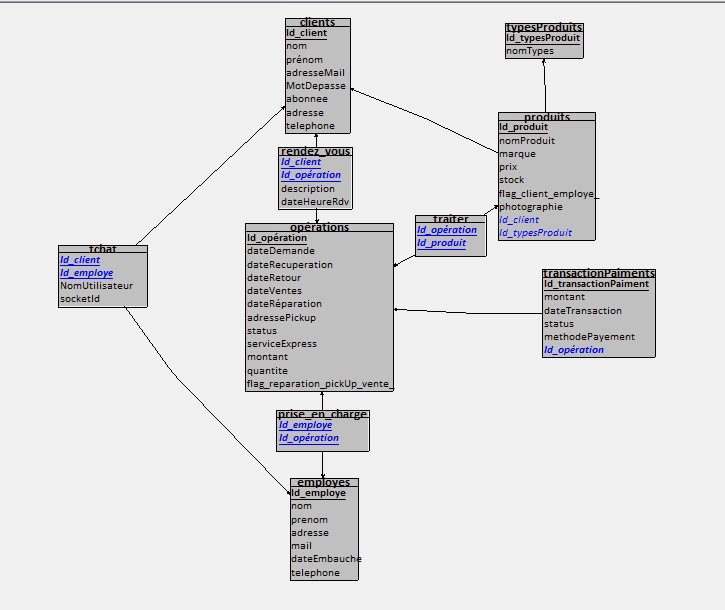
Définition des entités principales : clients, produits, opérations, rendez-vous, employés, transactions de paiement et types de produits.

Relations entre les entités : un client peut avoir plusieurs opérations, un produit peut appartenir à un type de produit, etc..



* **Modèle Logique de Données (MLD) :**

Schéma détaillé des tables et des relations de la base de données. Inclusion des clés étrangères et des contraintes pour assurer l'intégrité des données.



**4.2. Base de Données Distante (MySQL)**

La base de données distante sera utilisée pour stocker les informations principales et assurer la synchronisation avec la base de données locale de l'application.

**5. Phase de Développement**

**5.1. Prise en Main de React Native CLI**

**Choix de React Native CLI plutôt qu'Expo :**

React Native CLI et Expo sont deux options populaires pour développer des applications mobiles avec React Native. Cependant, pour le projet Caluire Mobile, nous avons choisi d'utiliser React Native CLI pour les raisons suivantes :

1. **Contrôle et Flexibilité Accrus :**
   * **Accès aux Modules Natifs :** React Native CLI offre un accès direct aux modules natifs, permettant une intégration plus facile des bibliothèques spécifiques à la plateforme (iOS et Android). Cela est particulièrement utile pour les fonctionnalités nécessitant des capacités natives avancées.
   * **Personnalisation du Code Natif :** Avec React Native CLI, il est possible de personnaliser et de modifier le code natif directement, ce qui permet une plus grande flexibilité dans le développement et l'optimisation de l'application.
2. **Dépendances et Bibliothèques :**
   * **Support des Bibliothèques Non-Supportées par Expo :** Certaines bibliothèques populaires ne sont pas supportées par Expo ou nécessitent des modules natifs qui ne sont pas inclus par défaut. React Native CLI permet l'installation et l'utilisation de n'importe quelle bibliothèque, sans restrictions.
3. **Taille de l'Application :**
   * **Optimisation de la Taille de l'Application :** Les applications développées avec React Native CLI peuvent être optimisées pour être plus légères, car elles n'incluent que les dépendances nécessaires. Expo, en revanche, inclut de nombreuses fonctionnalités par défaut, augmentant la taille de l'application finale.
4. **Délai de Compilation et de Déploiement :**
   * **Builds Personnalisés :** React Native CLI permet de créer des builds personnalisés pour différentes configurations de l'application, optimisant ainsi les performances et les délais de déploiement.
5. **Gestion des Versions :**
   * **Contrôle des Mises à Jour :** Avec React Native CLI, les développeurs ont un contrôle total sur les mises à jour des versions de l'application, ce qui permet une gestion plus précise des dépendances et des correctifs.

**Inconvénients de l'Expo :**

* **Limitations en termes de Modules Natifs :** Bien qu'Expo simplifie le développement en offrant de nombreuses fonctionnalités prêtes à l'emploi, il peut être restrictif lorsque des modules natifs spécifiques sont nécessaires.
* **Taille et Performance :** Les applications créées avec Expo peuvent être plus volumineuses et moins performantes en raison des nombreuses fonctionnalités incluses par défaut.
* **Flexibilité :** La flexibilité est réduite, car certaines configurations avancées ne sont pas possibles ou nécessitent de détacher (eject) l'application d'Expo, ce qui peut compliquer le processus de développement.

Pour ces raisons, React Native CLI a été choisi pour le développement de l'application Caluire Mobile, offrant ainsi une flexibilité, une personnalisation et une optimisation accrues pour répondre aux besoins spécifiques du projet.

**5.2. Mise en Place des Patterns**

Les patterns MVVM (Model-View-ViewModel) et Service Locator seront utilisés pour structurer l'application. Des services seront créés pour accéder aux données et communiquer avec l'API.

**5.3. Développement des Fonctionnalités**

Les principales fonctionnalités de l'application incluent :

* Authentification et gestion des comptes utilisateurs.
* Navigation et recherche de produits.
* Gestion des produits et de l'inventaire.
* Paiement sécurisé.
* Chat en temps réel pour le support client.
* Prise de rendez-vous pour la réparation et le pickup des produits.

**6. Utilisation de Mailjet pour l'Envoi de Mails**

**6.1. Configuration de Mailjet**

Mailjet est une plateforme d'emailing qui permet l'envoi et la gestion des emails transactionnels et marketing. Pour intégrer Mailjet à l'application Caluire Mobile, les étapes suivantes sont nécessaires :

1. **Création d'un Compte Mailjet :**
   * Inscription sur [Mailjet](https://www.mailjet.com/).
   * Configuration des paramètres du compte et génération des clés API (clé publique et clé privée).
2. **Installation du SDK Mailjet :**
   * Ajout de la dépendance Mailjet dans le projet .NET :

bash

Copier le code

dotnet add package Mailjet.Api --version 1.1.3

1. **Configuration des Clés API :**
   * Stockage des clés API dans le fichier de configuration de l'application (par exemple, **appsettings.json**).

**6.2. Envoi de Mails avec Mailjet**

Pour envoyer des emails avec Mailjet, un service d'email sera créé. Voici un exemple d'implémentation en C# :

**EmailService.cs :**

csharp

Copier le code

using Mailjet.Client; using Mailjet.Client.Resources; using Newtonsoft.Json.Linq; using System.Threading.Tasks; public class EmailService { private readonly string \_apiKey; private readonly string \_apiSecret; public EmailService(IConfiguration configuration) { \_apiKey = configuration["Mailjet:ApiKey"]; \_apiSecret = configuration["Mailjet:ApiSecret"]; } public async Task SendEmailAsync(string toEmail, string subject, string body) { MailjetClient client = new MailjetClient(\_apiKey, \_apiSecret); MailjetRequest request = new MailjetRequest { Resource = Send.Resource, } .Property(Send.FromEmail, "your-email@example.com") .Property(Send.FromName, "Caluire Mobile") .Property(Send.Subject, subject) .Property(Send.HtmlPart, body) .Property(Send.Recipients, new JArray { new JObject { {"Email", toEmail} } }); MailjetResponse response = await client.PostAsync(request); if (response.IsSuccessStatusCode) { Console.WriteLine("Email sent successfully."); } else { Console.WriteLine($"Error sending email: {response.GetErrorMessage()}"); } } }

**6.3. Vérification du Compte Client**

Lors de la création d'un compte client, un email de vérification sera envoyé pour confirmer l'adresse email du client. Voici le processus :

1. **Création d'un Token de Vérification :**
   * Génération d'un token unique pour chaque client lors de son inscription.
   * Stockage du token dans la base de données associé au client.
2. **Envoi de l'Email de Vérification :**
   * Utilisation du service d'email pour envoyer un lien de vérification contenant le token au client.
3. **Vérification du Token :**
   * Lorsque le client clique sur le lien, le token est envoyé au serveur.
   * Le serveur vérifie le token et active le compte du client si le token est valide.

**Exemple de Code pour la Vérification du Compte :**

**ClientsService.cs :**

csharp

Copier le code

public async Task RegisterClientAsync(Client client) { // Génération du token de vérification client.VerificationToken = Guid.NewGuid().ToString(); // Enregistrement du client dans la base de données await \_context.Clients.AddAsync(client); await \_context.SaveChangesAsync(); // Envoi de l'email de vérification string verificationUrl = $"https://your-app.com/verify?token={client.VerificationToken}"; string emailBody = $"<p>Merci de vous être inscrit. Veuillez vérifier votre compte en cliquant sur le lien suivant : <a href='{verificationUrl}'>Vérifier mon compte</a></p>"; await \_emailService.SendEmailAsync(client.Email, "Vérification de votre compte", emailBody); } public async Task<bool> VerifyClientAsync(string token) { var client = await \_context.Clients.FirstOrDefaultAsync(c => c.VerificationToken == token); if (client != null) { client.IsVerified = true; client.VerificationToken = null; await \_context.SaveChangesAsync(); return true; } return false; }

**7. Synchronisation et Sécurité**

**7.1. Synchronisation des Données**

Un service de synchronisation sera mis en place pour assurer la cohérence entre la base de données locale et la base de données centrale. Les API RESTful seront utilisées pour la communication entre le front-end et le back-end.

**7.2. Sécurité des Données**

Les protocoles SSL seront utilisés pour sécuriser les transactions. Des mécanismes de cryptage seront mis en place pour protéger les données sensibles, comme les mots de passe et les informations de paiement.

**8. Conclusion**

Le développement de l'application Caluire Mobile vise à offrir une plateforme pratique et sécurisée pour l'achat de produits et la gestion des réparations. En utilisant des technologies modernes et des pratiques de développement sécurisées, l'application garantira une expérience utilisateur optimale.

**9. Annexes**

**Diagramme MCD :**

**Diagramme MLD :**

**SQL Schema :** Le fichier SQL **caluiremobile.sql** contient la création des tables et des relations dans la base de données.

**Exemples de Code**