**Cahier des charges : Application mobile de recherche de restaurants**

**1. Présentation du projet**

Le projet consiste en la création d'une application mobile pour la recherche de restaurants à proximité, développée en **React Native** avec **Expo**. L’application a pour but d'aider les utilisateurs à découvrir des restaurants autour d'eux, en fournissant des informations détaillées sur chaque établissement, des offres promotionnelles, et des options de navigation et de contact. Elle vise à se démarquer par des fonctionnalités innovantes telles que les suggestions basées sur les préférences de l’utilisateur et des recommandations personnalisées grâce à l'IA.

**2. Objectifs du projet**

* Offrir une expérience utilisateur intuitive pour la découverte de restaurants.
* Intégrer des fonctionnalités de géolocalisation pour localiser les établissements à proximité.
* Fournir des suggestions et recommandations basées sur les préférences de l’utilisateur.
* Mettre en place un système de notifications et de promotions.
* Offrir une fonctionnalité de chat avec les restaurants (en option pour un usage ultérieur).

**3. Fonctionnalités principales**

**3.1 Fonctionnalités utilisateur**

* **Page de bienvenue (Welcome Page)** : Affiche lors du premier lancement de l'application.
* **Pages de connexion et d'inscription** : Authentification des utilisateurs pour accéder à des fonctionnalités personnalisées.
* **Page d'accueil (Home)** : Affiche la liste des restaurants proches de l'utilisateur avec des options de filtrage et de tri.
* **Recherche avancée** : Permet à l’utilisateur de rechercher des restaurants par nom, emplacement, type de cuisine, etc.
* **Détails des restaurants** : Pour chaque restaurant, affichage de photos, adresse, horaires, avis et un bouton pour obtenir l’itinéraire.
* **Notifications et promotions** : Notifications contextuelles pour informer l'utilisateur des promotions et des nouveaux restaurants.
* **Carte interactive** : Intégration de **TomTom Maps** pour afficher les restaurants sur une carte avec des options de recherche et d'itinéraire.
* **Chat avec les restaurants** : Option de chat avec les établissements pour poser des questions ou réserver.
* **Profil utilisateur** : Écran de profil avec options pour modifier les informations, gérer les paramètres de notifications, et accéder aux préférences de langue et de localisation.

**3.2 Fonctionnalités administratives (Backend)**

* **Gestion des restaurants** : Ajouter, mettre à jour et supprimer les informations des restaurants.
* **Gestion des promotions** : Publier et modifier des offres promotionnelles spécifiques à chaque restaurant.
* **Système de notifications** : Envoyer des notifications en fonction des préférences et de la localisation de l'utilisateur.
* **Analyse des préférences** : Utiliser des modèles d'IA pour analyser les préférences de l’utilisateur et fournir des recommandations personnalisées.

**4. Architecture de l'application**

**4.1 Frontend**

* **Framework** : React Native avec Expo.
* **Langage** : TypeScript avec des fichiers .tsx.
* **Interface utilisateur** : Utilisation de composants react-native-paper et un design responsive.

**4.2 Backend**

* **Technologies** :
  + **Express**  pour créer une API REST pour gérer les données des restaurants.
  + **Base de données** : SQLite pour une version légère ou PostgreSQL pour une échelle plus grande.
  + **Gestion des utilisateurs et des sessions** : Authentification JWT.
* **Modules IA** : Modèles de recommandations et préférences, opérant hors ligne sur le mobile de l'utilisateur.

**5. Design et UX**

* **Thème de l'application** : Interface simple et moderne avec un thème clair et sombre.
* **Expérience utilisateur** : Navigation intuitive avec une présentation claire des restaurants et des notifications informatives.

**6. Technologies et outils**

* **Géolocalisation et Cartographie** : Google Maps pour la carte interactive.
* **Backend en NodeJS** : **Express** pour les données des restaurants.
* **Stockage de données local** : **AsyncStorage** pour stocker l’état de démarrage de l’application.
* **Authentification** : **JWT** pour les sessions utilisateurs et le chat.

**7. Contraintes et limitations**

* **Fonctionnement hors ligne** : La suggestion d’IA doit fonctionner en mode hors connexion.
* **Performances** : L'application doit être fluide avec des temps de chargement réduits pour garantir une bonne expérience utilisateur.
* **Confidentialité** : Les données personnelles doivent être sécurisées.

**8. Planification du développement**

**Phase 1 : Analyse et conception**

* Étude des exigences.
* Conception des écrans principaux et des fonctionnalités.

**Phase 2 : Développement de l'interface utilisateur**

* Implémentation des pages de bienvenue, de connexion, d’inscription, et du profil utilisateur.
* Affichage de la liste de restaurants et détails des restaurants.

**Phase 3 : Intégration du backend et de la base de données**

* Création de l'API en Flask/FastAPI.
* Configuration de la base de données SQLite/PostgreSQL.

**Phase 4 : Intégration des fonctionnalités avancées**

* Intégration de la carte avec TomTom.
* Développement de la logique de notifications.
* Déploiement des modèles d’IA pour les recommandations.

**Phase 5 : Tests et déploiement**

* Tests unitaires et d’intégration.
* Optimisation de l’application.
* Déploiement final sur les plateformes mobiles.

**9. Suivi et maintenance**

Après le lancement, prévoir des mises à jour pour :

* Corriger les bugs signalés par les utilisateurs.
* Ajouter des fonctionnalités supplémentaires en fonction des retours.
* Adapter l'application aux évolutions d'Expo et des systèmes d'exploitation.

**Modèle Conceptuel de Données (MCD)**

Pour le MCD, chaque entité est représentée avec ses attributs, et les relations sont illustrées pour refléter les liens entre les entités :

1. **Restaurant** (0,n) ----(1,n)---- **Promotion**
2. **Restaurant** (0,n) ----(1,1)---- **Menu**
3. **Restaurant** (1,n) ----(1,n)---- **Réservation** ----(0,1)---- **Utilisateur**
4. **Restaurant** (1,n) ----(1,n)---- **Avis** ----(0,1)---- **Utilisateur**

**Modèle Logique de Données (MLD)**

Ce MLD repose sur la structure que tu as décrite, avec les types de données et les clés étrangères pour chaque entité.

**1. Table Restaurant**

* ID : INTEGER (clé primaire, auto-incrémentée)
* Nom : VARCHAR(255) (nom du restaurant)
* Latitude : FLOAT
* Longitude : FLOAT
* PlacesReservation : INTEGER
* TypeCuisine : VARCHAR(255) (par exemple, liste de types séparés par des virgules)

**2. Table Menu**

* ID : INTEGER (clé primaire, auto-incrémentée)
* RestaurantID : INTEGER (clé étrangère vers Restaurant(ID))
* NomPlat : VARCHAR(255)
* Description : TEXT
* Prix : FLOAT
* Categorie : VARCHAR(50) (entrée, plat, dessert)

**3. Table Promotion**

* ID : INTEGER (clé primaire, auto-incrémentée)
* RestaurantID : INTEGER (clé étrangère vers Restaurant(ID))
* Description : TEXT
* DateDebut : DATETIME
* DateFin : DATETIME

**4. Table Utilisateur**

* ID : INTEGER (clé primaire, auto-incrémentée)
* Nom : VARCHAR(255)
* Email : VARCHAR(255) (unique)
* MotDePasse : VARCHAR(255) (mot de passe hashé)

**5. Table Réservation**

* ID : INTEGER (clé primaire, auto-incrémentée)
* RestaurantID : INTEGER (clé étrangère vers Restaurant(ID))
* UtilisateurID : INTEGER (clé étrangère vers Utilisateur(ID))
* DateReservation : DATETIME
* PlacesReservees : INTEGER

**6. Table Avis**

* ID : INTEGER (clé primaire, auto-incrémentée)
* RestaurantID : INTEGER (clé étrangère vers Restaurant(ID))
* UtilisateurID : INTEGER (clé étrangère vers Utilisateur(ID))
* Note : INTEGER (entre 1 et 5)
* Commentaire : TEXT
* DateAvis : DATETIME