Project Document: Roommate Finder Application

Overview

The **Roommate Finder Application** is designed to help users find compatible roommates based on a questionnaire. The app will match users by calculating a compatibility percentage from their answers. It uses **Firebase** for user authentication and storing data, while the app itself is built using **Flutter** and follows the **MVVM** (Model-View-ViewModel) architecture.

Project Structure

- **Frontend**: Mobile application built with **Flutter**.
- **Backend**: Powered by **Firebase** for managing user data and authentication.
- **Architecture**: **MVVM** design pattern.
- **Repositories**: Act as a bridge between data sources and UI.

Features

1. **User Authentication**
    - Sign up
    - Sign in
    - Password recovery
2. **Room Listings**
    - List all available rooms
    - Room details screen
3. **Questionnaire**
    - Compatibility questionnaire
    - Weighting mechanism for answers
    - Percentage calculation for compatibility
4. **User Profiles**
    - User details
    - User ratings and reviews
5. **Room Reviews**
    - User reviews for rooms
    - Rating system for rooms

Team Structure

- **Tasneem**: Frontend Developer (Room Listings & Room Details)
- **Amira**: Frontend Developer (Questionnaire & User Profiles)
- **Lobna**: Backend Developer (Firebase Integration & Authentication)
- **Ethar**: Backend Developer (Repositories & Data Sources)
- **Yassen**: Frontend Developer (Reviews & Ratings)
- **Mazen**: Full-Stack Developer (Integration & Testing)

Task Assignments

Feature	Task	Developer(s)	Description
User Authentication	**Task 1.1**: Implement Firebase Authentication	Tasneem, Amira, Lobna	- **Tasneem**: Set up Firebase project and integrate with Flutter app. - **Amira**: Implement sign-up using `firebase_auth`. - **Lobna**: Implement sign-in and password recovery functionality.
	Task 1.2: Create user profiles in Firestore	Lobna	Store user details (name, email, profile picture) in Firestore.
Room Listings	**Task 2.1**: Implement Room List Screen	Tasneem, Ethar	- **Tasneem**: Create UI layout for listing rooms using Flutter widgets. - **Ethar**: Fetch room data from Firestore and display it in a `ListView`.
	Task 2.2: Implement Room Details Screen	Mazen, Tasneem	- **Mazen**: Create detailed room view UI. - **Tasneem**: Display room info (price, location) and add a button to view reviews.
Questionnaire	**Task 3.1**: Design Questionnaire UI	Amira, Mazen	- **Amira**: Create a list of questions using `Form` and `TextField` widgets. - **Mazen**: Implement UI to collect answers.
	Task 3.2: Implement Weighting System	Ethar	Define weights for each answer and store questionnaire results in Firestore.
Compatibility Calculation	**Task 4.1**: Implement Compatibility Logic	Lobna	Write a function to calculate compatibility between users based on their answers.
	Task 4.2: Store Compatibility Results	Mazen, Ethar	- **Mazen**: Save compatibility scores in Firestore. - **Ethar**: Ensure the data is stored properly.
User Profiles and Reviews	**Task 5.1**: Implement Profile Screen	Amira, Lobna	- **Amira**: Display user details (name, bio) and allow editing. - **Lobna**: Fetch and display profile data from Firestore.
	Task 5.2: Implement User Reviews Feature	Yassen, Amira	- **Yassen**: Build UI to submit reviews. - **Amira**: Collaborate on displaying reviews.

Feature	Task	Developer(s)	Description
Room Reviews and Ratings	**Task 6.1**: Implement Room Review System	Yassen, Tasneem	- **Yassen**: Build UI to submit room reviews. - **Tasneem**: Connect review submission with room listings.
	Task 6.2: Display Room Ratings	Mazen, Yassen	- **Mazen**: Calculate average ratings and display them on the room details page. - **Yassen**: Design the UI to show ratings.

Folder Structure

```
roommate-finder-app/
├── android/                # Android-specific files
├── ios/                    # iOS-specific files
├── lib/                    # Main application code
│   ├── models/             # Data models
│   ├── viewmodels/         # ViewModels for MVVM architecture
│   ├── views/              # UI components and screens
│   │   ├── authentication/ # Authentication screens
│   │   ├── listings/       # Room listings screens
│   │   ├── questionnaire/  # Questionnaire screens
│   │   ├── profiles/       # User profile screens
│   │   ├── reviews/        # Reviews and ratings screens
│   ├── services/           # Services for data fetching
│   ├── repositories/       # Repositories for data management
│   ├── utils/              # Utility classes and functions
│   └── main.dart           # Main entry point of the app
├── test/                   # Unit and widget tests
├── pubspec.yaml            # Project dependencies
└── README.md               # Project documentation
```

Notes

Category	Notes
Firebase Notes	- **Firebase Auth** is a ready-made solution for user authentication. You don't need to code the authentication process from scratch. The `firebase_auth` package in Flutter simplifies tasks like sign-in, sign-up, and password recovery. - **Firestore** is a NoSQL database provided by Firebase that allows easy data storage. When you're working with data (like room listings, user profiles), Firestore will help you store, retrieve, and update it.

Category	Notes
Project Structure Notes	- **MVVM Architecture**: This pattern separates the code into **Model** (data), **View** (UI), and **ViewModel** (logic connecting data and UI). It's used to make the code more organized and easier to maintain. - **Repository**: It's a layer in between your Firebase data and the ViewModel that helps manage how data is retrieved and sent. Think of it as a "middleman."
Folder Structure Notes	- **lib/models**: Contains classes representing data (e.g., user profiles, room details). - **lib/viewmodels**: This is where the app logic lives. It connects data (from models) and the user interface (views). - **lib/views**: Holds all the UI (user interface) code, organized by different screens (like login, room listings). - **lib/repositories**: Acts as a bridge between Firestore and the ViewModels.
Additional Hints	- **Questionnaire Task**: Use `Form` and `TextField` widgets in Flutter to gather user inputs for the questionnaire. You can easily add validation to these fields (e.g., required answers) using `Validator` functions in Flutter forms. - **Room Listings Task**: Utilize `ListView.builder` for efficiently displaying a large number of room listings. This widget lazily builds its children, which improves performance. - **User Profiles Task**: Use `StreamBuilder` to listen to real-time updates from Firestore and reflect changes in the user profile screen immediately. - **Room Reviews Task**: Implement a `RatingBar` widget for users to submit their ratings. This can be found in the `flutter_rating_bar` package.

Resources

- Firebase Docs (Official guide to Firebase with Flutter)
- Flutter Firebase Setup (Setting up Firebase for Flutter)
- Firebase Authentication
- Firebase Firestore
- Firebase Storage
- Firebase crash course
- Beginner's Flutter Course (Arabic)
- Beginner's Firebase Course (Arabic)
- MVVM Beginner's Course (Arabic)
- MVVM Tutorial Repo

Project Repository

- Main Repository
- Task 1 Branch
- Task 2 Branch