**Developer Guide**



**LuxuryStay Hospitality**

|  |  |
| --- | --- |
| **Student1332408** | **MUHAMMAD USMAN TARIQ** |
| **Student1207064** | **AARSH KHAN** |
| **Student1375259** | **FARIHA MUSTAFA** |
| **Student1373398** | **ABDULLLAH AHMED** |
| **Student1298973** | **MUHAMMAD TAHA** |

**Requirements:**

**MERN (MongoDB, ExpressJS, React, Node.JS):**

* Proficiency in React for building dynamic user interfaces and handling state management.
* Ability to design and develop responsive UI components using React components and libraries like React Router, Material-UI, etc.
* Strong experience with Node.js and Express.js for building RESTful APIs, handling server-side logic, and managing backend functionality.
* Familiarity with MongoDB for database management, data modeling, and integrating it with backend services.

**State Management:**

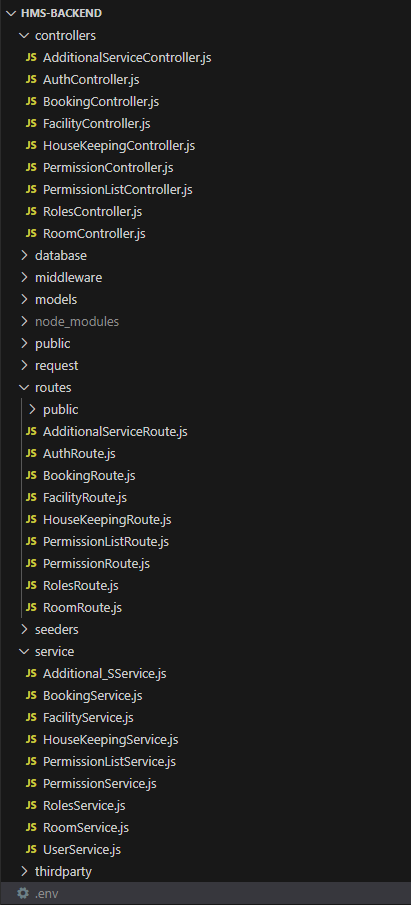
• Experience with React Context API for global state management.

**MongoDB Integration:**

• Knowledge of integrating MongoDB for database management.

• Implementing MongoDB Atlas for cloud database hosting and synchronization.

* This developer guide consists all the screenshots of the relevant controllers, services and paths with their relevant description for developers.



* **Source Code Structure divided into five folders:**

1. **Controllers**

This folder contains all of the relevant controllers with various CRUD operations which we’re using for the API multiple times.

1. **Midddleware**

The Controllers folder contains all the relevant controller which we’re using to manage state between different screens and also performing CRUD operations with the database.

1. **Pages**

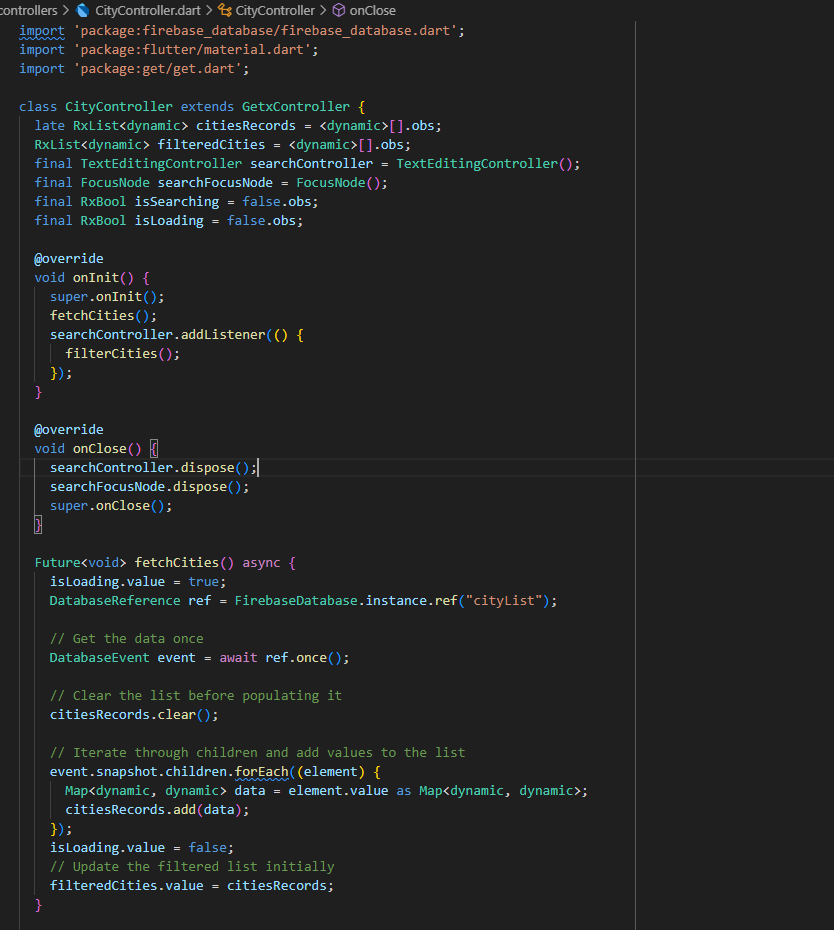
The pages folder contains (UI) all the screen widgets displaying data to the users.

1. **Services**

Services folder contains the push notifications file which is created to organize receiving notifications in all three on Background, on Foreground and on Terminated state notifications.

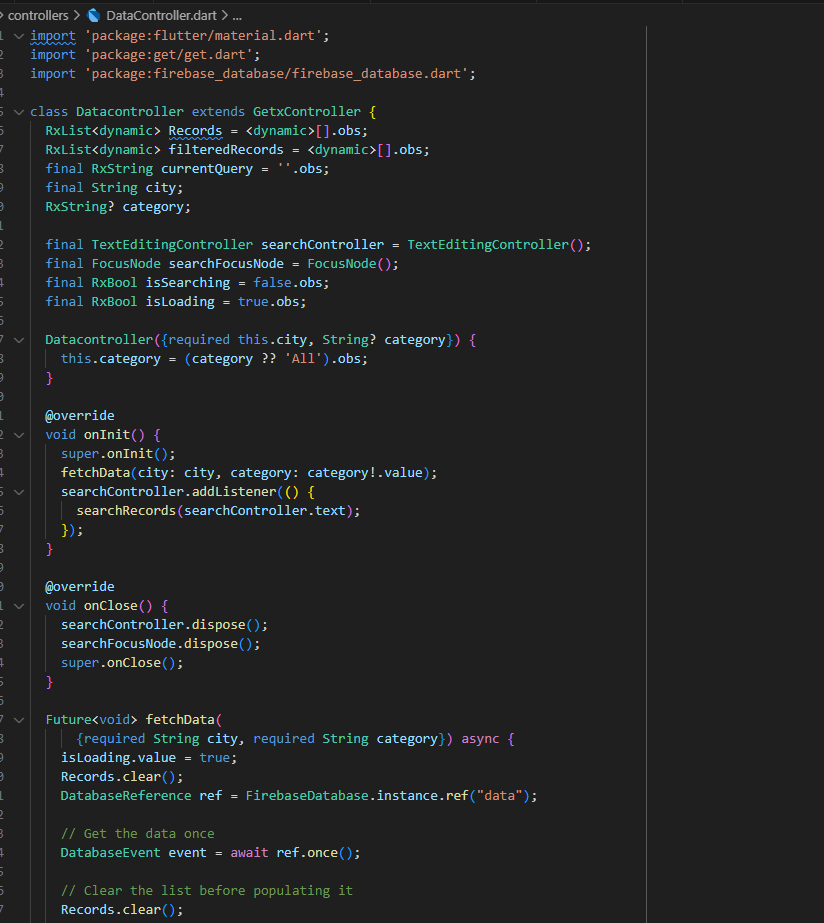
1. **Theme**

Theme folder contains the ColorTheme class used to implement default theme colors across the code.



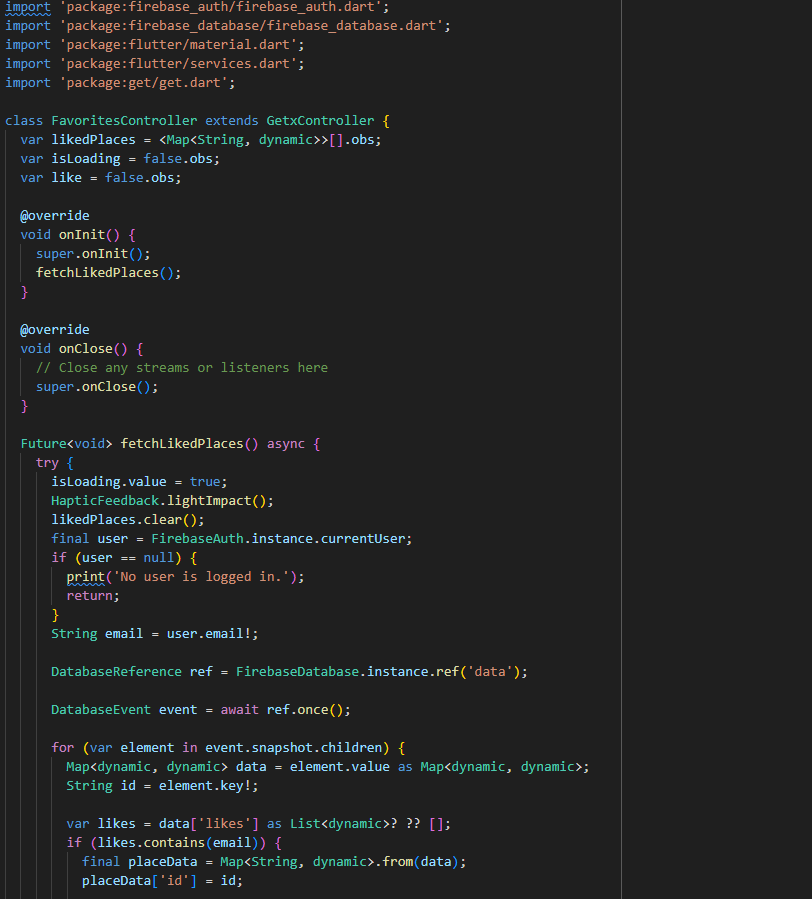
**City Controller:**

* Manages city data fetched from Firebase Realtime Database.
* Uses GetX for reactive state management.
* Handles data fetching, filtering based on user search input, and updates UI accordingly.
* Ensures efficient resource management with initialization and cleanup methods.



**Data controller:**

* Manages data retrieval from Firebase based on city and category.
* Uses GetX for reactive state management.
* Computes average ratings asynchronously.
* Supports sorting data by rating and searching based on user queries.
* Provides efficient initialization and cleanup of resources.



**Favorites Controller:**

* Manages user's liked places using Firebase Realtime Database.
* Utilizes GetX for reactive state management.
* Fetches and displays liked places based on current user.
* Calculates average ratings asynchronously for each liked place.
* Allows users to like/unlike places and updates database and UI accordingly.



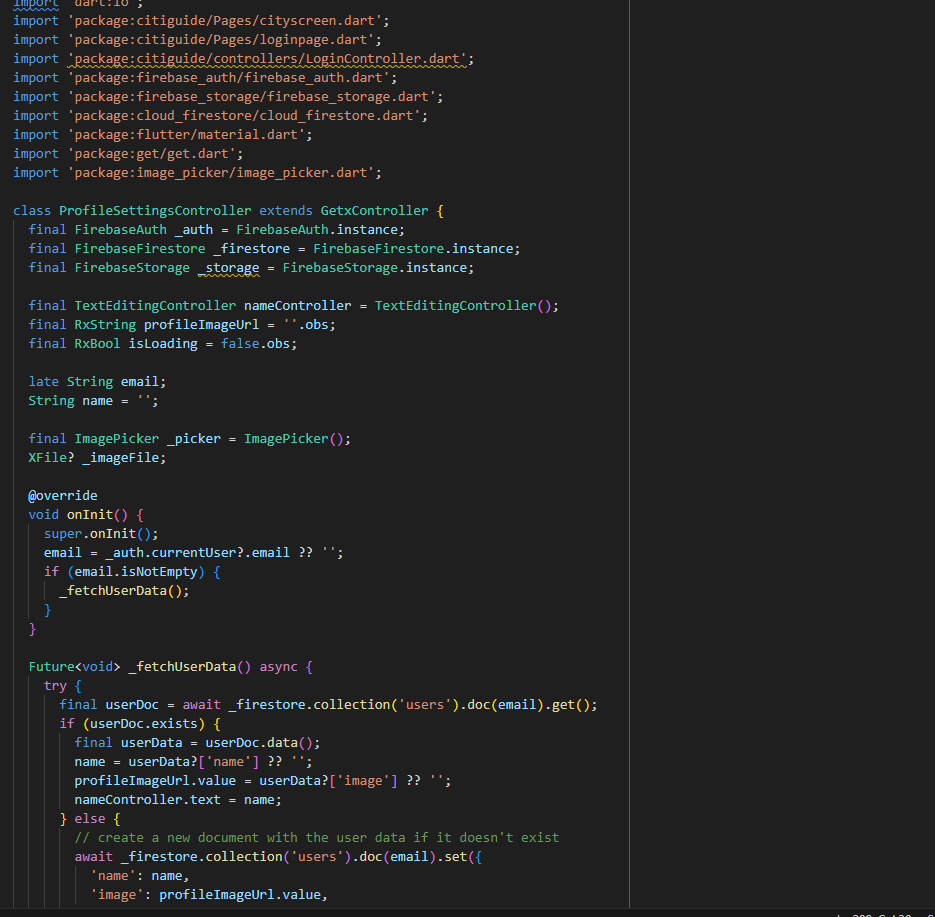
**Login Controller:**

* Manages user authentication using Firebase Authentication.
* Utilizes GetX for reactive state management.
* Handles sign-in with email and password, displaying appropriate error messages for different scenarios.
* Clears input fields and manages loading state during sign-in process.
* Provides methods for sign-out, sending password reset emails, and navigates between screens using GetX navigation.



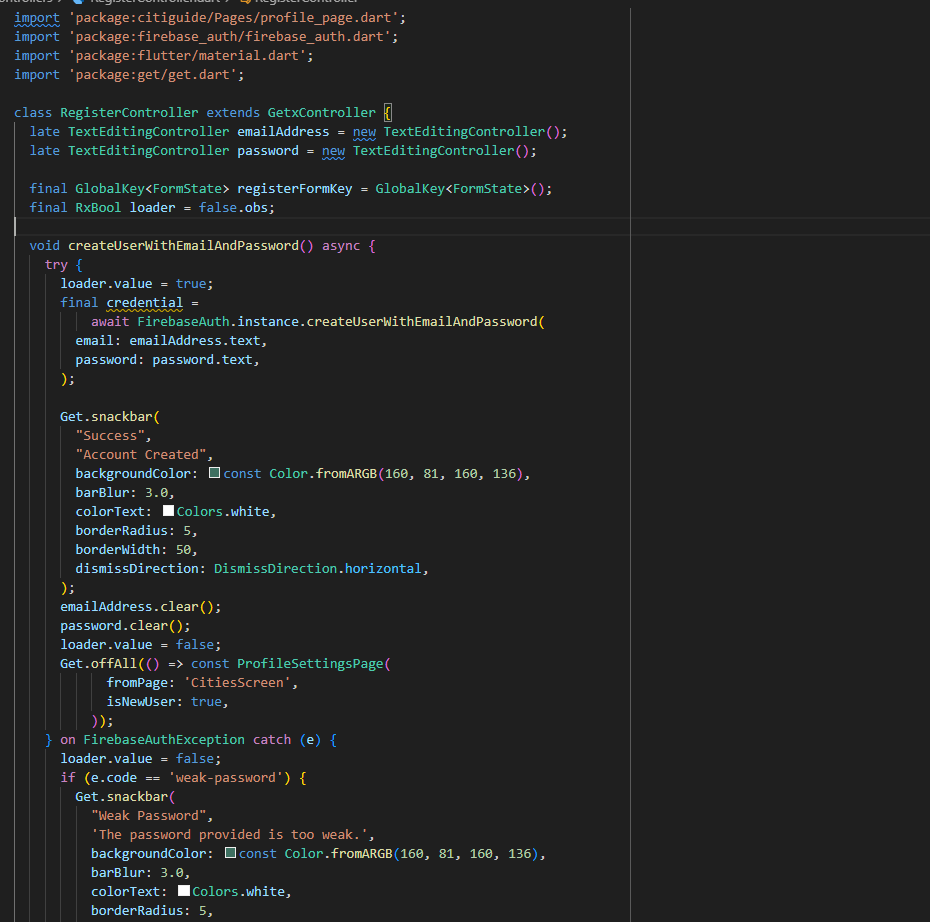
**Notification Controller:**

* Manages notifications using Firebase Firestore.
* Utilizes GetX for reactive state management.
* Fetches notifications for the current user and listens for real-time updates using Firestore snapshots.
* Allows adding, deleting, and marking notifications as read.
* Updates unread notification count based on the current state.



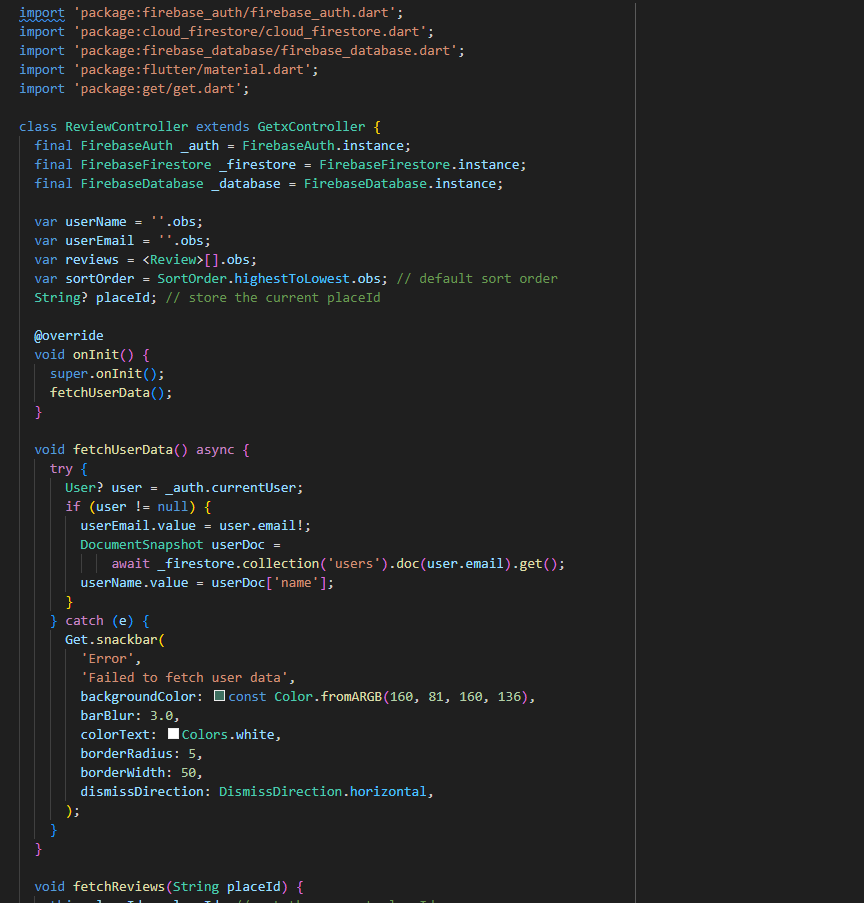
**Profile Settings Controller:**

* Manages user profile settings using Firebase Authentication, Firestore, and Storage.
* Utilizes GetX for reactive state management.
* Handles updating user name, changing password, uploading and displaying profile image, and deleting user account.
* Provides feedback to the user through snackbar messages for success and error scenarios.



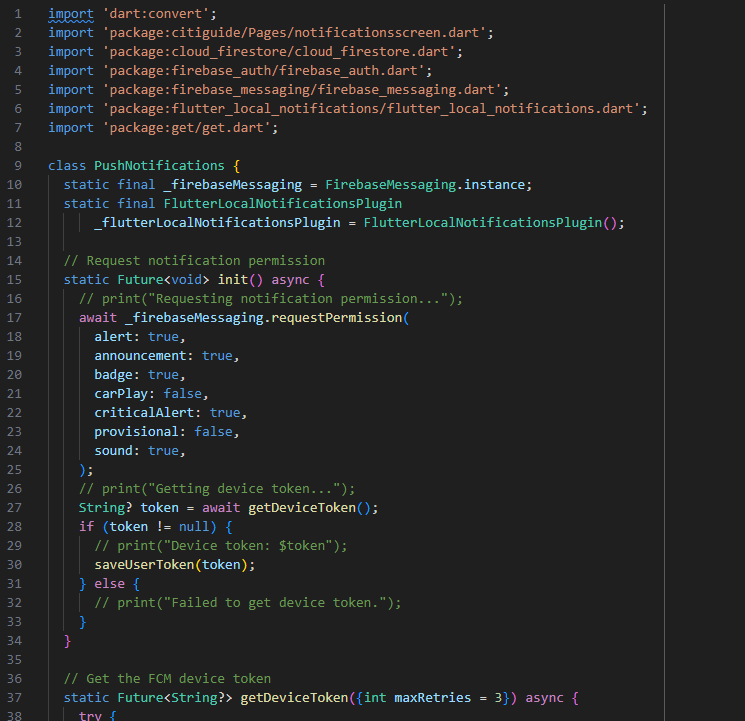
**Register Controller:**

* Handles user registration using Firebase Authentication.
* Utilizes GetX for reactive state management.
* Manages registration form validation and submission.
* Displays snackbar messages for success and common error scenarios during registration.



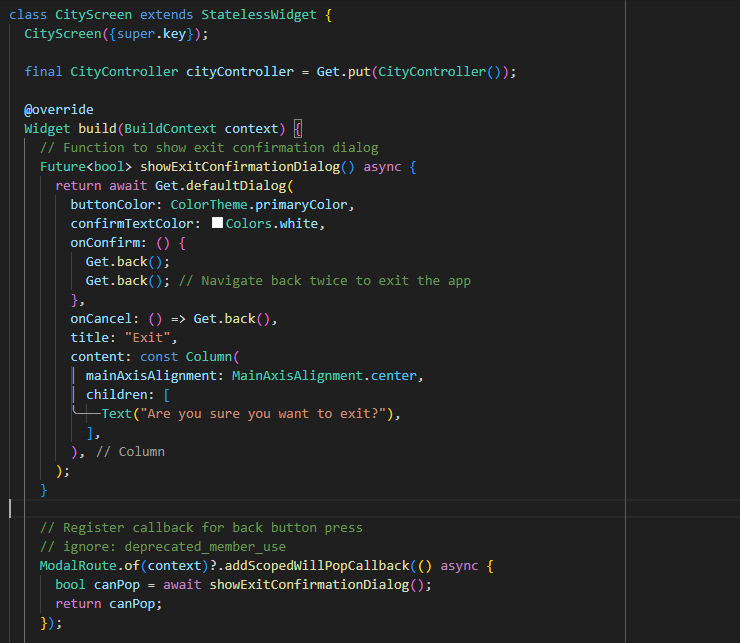
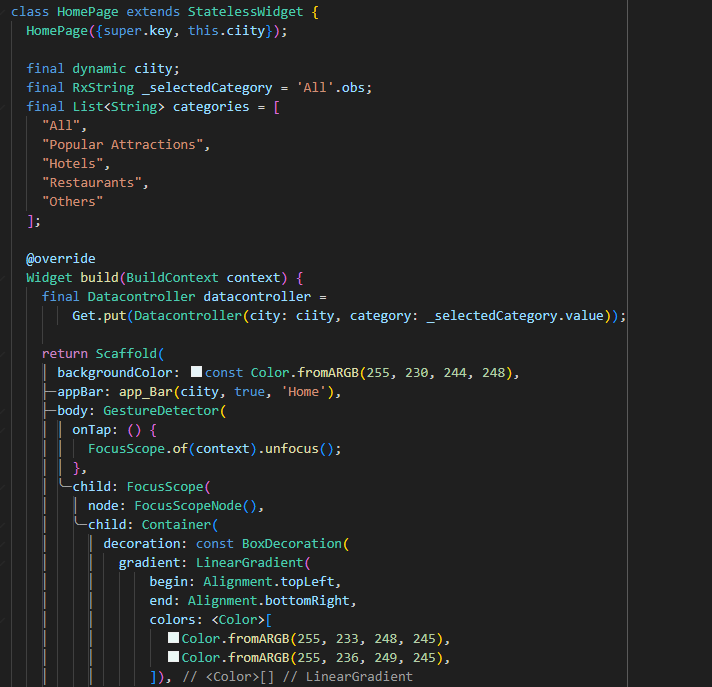
**Review Controller:**

* Manages user reviews using Firebase Authentication, Firestore, and Realtime Database.
* Utilizes GetX for reactive state management.
* Fetches user data (name and email) and reviews associated with a specific place.
* Allows users to add reviews with rating and comments, ensuring each user can only review a place once.
* Provides functionality to toggle between sorting reviews from highest to lowest rating and vice versa.



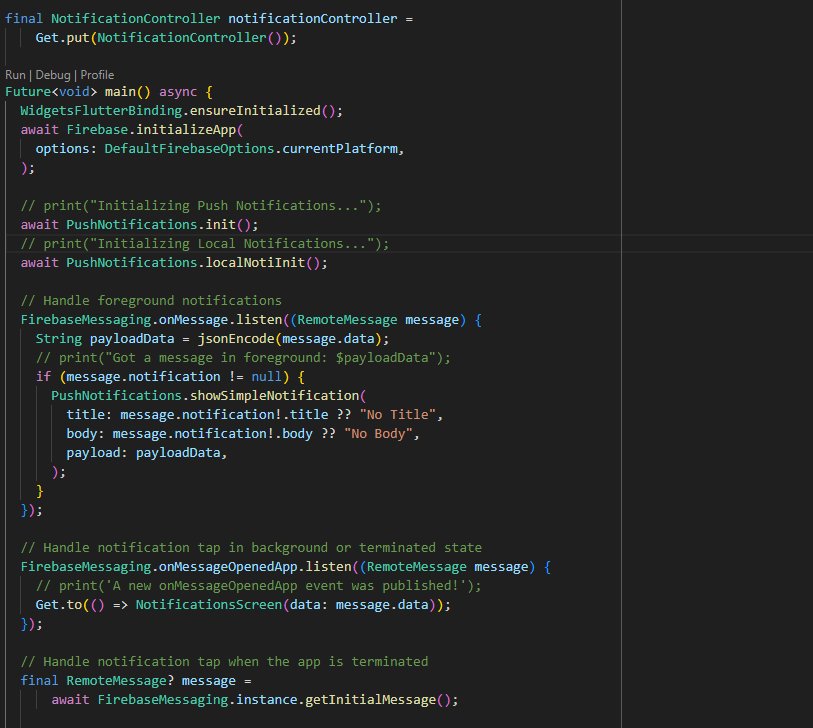
**Push Notifications class:**

* Manages push notifications using Firebase Cloud Messaging (FCM) and local notifications.
* Utilizes FlutterLocalNotificationsPlugin for displaying local notifications.
* Requests notification permissions and retrieves the device token for FCM.
* Saves the FCM token to Firestore under the current user's document.
* Checks if a token already exists for another user before saving.
* Handles tapping on notifications to navigate to the NotificationsScreen and display additional data.

**Pages (UI) folder:**

* This folder contains all the UI screens of the application, each coupled with its corresponding controller for managing state and business logic.
* Developers can quickly locate and work on specific screens, enhancing productivity during development and troubleshooting.
* Screens are categorized logically based on their functionalities or user flows.



**Main.dart Page:**

* Firebase Integration: Initializes Firebase services (Firebase.initializeApp()) to enable authentication and cloud messaging.
* Push Notifications: Uses PushNotifications class to manage FCM and local notifications (FlutterLocalNotificationsPlugin).
* Foreground Notifications: Listens for messages (FirebaseMessaging.onMessage) and displays notifications using PushNotifications.showSimpleNotification().
* Background Notifications: Handles notification taps (FirebaseMessaging.onMessageOpenedApp) and app launches with notification data (FirebaseMessaging.getInitialMessage()).
* User Authentication: Checks user authentication status (FirebaseAuth.currentUser) to determine the initial screen (WelcomeScreen, LoginPage, or CityScreen).
* Navigation Management: Utilizes GetX (Get.to()) for seamless navigation to NotificationsScreen based on notification data, ensuring context preservation.
* Theme Customization: Configures ThemeData for consistent app styling, enhancing user interface aesthetics and usability.
* State Management: Implements reactive state management (GetMaterialApp) for efficient UI updates based on changes in authentication status and notification reception.

END