ICCS 448 - Mobile App - Final Project Report

Name: Haicheng Wang

Student ID: 6580244

project name: Nearby Explorer

Project Description

Nearby Explorer is a Flutter-based mobile app that helps users discover, explore, and interact with places of interest around them(The rank is done by highest rating score to the lowest). Leveraging the Google Places API for powerful nearby-search and text-search capabilities, it offers both list and map views with adjustable search radius and type filters. Users can tap into a rich place details page—photos, ratings, reviews—and comment to a community section with one-level nested replies, complete with real-time Firebase notifications when someone responds. This comment section is the thing shared among all users(but users still can choose comments anonymously to hide their display name and avatars), and what is private to each user is that users can favorite places for quicker access next time. Users also can manage their profile information, and enjoy a guided, animated onboarding experience. Under the hood, Firebase Firestore stores all data with carefully crafted security rules to ensure privacy, users are able to edit their own comments only, they cannot delete other comments created by other accounts.

Login & Register Screen

In the login page, I defined two tabs one for “Log In” and one for “Register” that will navigate to the register screen. I used several Hero classes to move the decoration elements between these two screens to improve UX. Besides, once a user successfully registers one account, once coming back from the register screen, I push the screen with the newEmail to prefill the just registered account email, so users only need to type in the password again.

Under the hood, I make use of Firebase Authentication like we did in class to handle all credential checks, account creation, and password resets. As soon as the user taps the primary button (“Sign In” or “Register”), the entire form fades into a centered loading spinner until Firebase returns a response. Errors such as “Invalid email format,” “Wrong password,” or “Email already in use” are displayed inline beneath the offending field, and the form gently shakes and turns into red to draw the user’s attention back to unresolved issues.

Onboarding Screen

When a new user first opens the app, they're greeted by a three-step onboarding flow that feels both personal and engaging. The first page lets them "Make your profile" by choosing or uploading an avatar and typing a display name. Subtle SVG decorations slide and fade into view around the header. After they've set up their profile, a smooth page indicator and "Next" button guide them to a concise overview of the app's four main tabs ("Your Tabs"), where each icon fades and slides in sequence. Finally, they see a detailed list that pairs each tab's hero-animated icon with a brief description of its function—helping them understand Explore, Favorites, Notifications, and Profile before diving in.

Under the hood, I built this using a PageView controlled by a PageController and a SmoothPageIndicator for clear progress feedback. Avatar selection leverages image_picker for gallery input or a set of hosted defaults, while profile information is saved via FirebaseAuth (for displayName/photoURL) and SharedPreferences (to record that onboarding is complete). I store custom avatars in Firebase Storage. All animations—slide, fade, scale, even the icon "fly-to" transitions—are handled by the flutter_animate package.

Home Screen

The Home Screen serves as the central hub of Nearby Explorer, presenting four core areas—Notifications, Explore, Favorites, and Profile—behind a simple, bottom-navigation bar. At the top level, the screen displays only the currently selected page's content, keeping the UI clean and focused. Swiping is deliberately disabled here in favor of a clear tap-based interface: each BottomNavigation item is labeled and uses an intuitive icon, with the active tab highlighted in your primary theme color and the others rendered in a subdued gray. By default the Explore tab is selected, immediately inviting users to discover nearby places, but a quick tap on any other icon seamlessly switches to that feature's standalone module.

Under the hood, the Home Screen is a compact StatefulWidget that initializes a single PlaceService and assembles a List<Widget> of the four feature pages in initState. The Scaffold's body simply displays _pages[_currentIndex], and setState on the onTap callback of BottomNavigationBar updates _currentIndex, triggering an efficient rebuild of just the body. This modular structure keeps each feature in its own file—ExplorePage, NotificationsPage, FavoritesPage, and ProfilePage—so the Home Screen remains lean, easy to read, and simple to extend or reorder in the future.

Explore Screen

Upon entering the Explore screen, users are immediately centered on their current location, visualized either as a scrollable list of nearby places or as an interactive map. A persistent search bar at the top allows users to set a searchable place as new origin, while a row of choice-chips filters results by place type (e.g. restaurants, parks). Below, a slider adjusts the search radius in kilometers, and a toggle switches between list and map views. In the list view, each tile shows the place's name, star rating, and distance from the chosen origin. Tapping on each one will bring users to the place detail page.

Behind the scene, the page is a StatefulWidget that first requests location permission via the Geolocator plugin and captures the device's coordinates. It then calls a custom PlaceService—which wraps Google's Nearby and Text Search REST endpoints—to fetch and sort results by rating or distance. User input in the search bar is debounced (400 ms) to minimize API calls, and search results rebind the origin, triggering a refresh. The Google Maps Flutter widget renders circles and markers for the map view, while standard ListView.builder and ChoiceChip controls compose the list interface.

Favorites Screen

The Favorites screen presents a personalized, scrollable list of places the user has starred. A pink gradient background wraps the entire view, and each favorite appears as a semi-transparent card with a leading photo (or placeholder icon), the place's name, its star rating, and address. Tapping a card slides up a modal bottom sheet with full place details.

I leverage a StreamBuilder on the Firestore users/{uid}/favorites collection—ordered by timestamp—to drive real-time updates. Each document is mapped to a slim Place model. and rendered them also into the place detail page. The pink gradient is applied via a Container's BoxDecoration. Finally, I pulled in the flutter_animatepackage to stagger each card's entrance with fade, slide, and subtle scale animations, creating a polished, dynamic list.

Notifications Screen

Over a SafeArea, a live-updating list of reply notifications scrolls vertically. Each entry is a rounded tile showing the replier’s avatar, a headline like “Alice replied to your comment,” and a “Tap to view it” subtitle. Unread tiles carry a bold accent and “new” badge, while read ones fade into a softer white. Tapping any notification atomically marks it as read in Firestore and slides up the corresponding place detail page in a modal bottom sheet, keeping the UX fluid and in context.

Beneath the UI, a StreamBuilder listens to users/{uid}/notifications in Firestore, ordering by createdAt timestamp. On tap, I update the document’s read flag and reconstruct a minimal Place model to drive the detail sheet. Finally, I wire in flutter_animate on each ListTile to create fade-in and slide-up effects (with increasing delays).

Profile Screen

The Profile screen centers the current user’s avatar and name at the top. Tapping the avatar opens the device gallery for a new photo, or the user can pick one of four hosted defaults from a horizontally scrolling row beneath. Once a change is made, the “Save Avatar” button becomes active—tapping it uploads the image to Firebase Storage, updates the user’s photoURL, and shows a confirmation snackbar. Finally, a “Log Out” tile sits at the bottom in a gently shaded red box. Currently, editing the display name isn’t supported—only avatar changes can be made here.

I pull the signed-in user via FirebaseAuth.instance.currentUser, display their displayName or email fallback, and manage avatar state locally with a combination of a File (for gallery picks) or an index into our default-URL list. Also I defined animation for fade-and-slide effects on each section. When saving, either putFile the picked image to avatars/{uid}.jpg in Firebase Storage or reuse a default URL, then call user.updatePhotoURL() and user.reload(). Snackbars communicate success or failure. Name changes are intentionally omitted to avoid the extra validation and edge-cases involved in renaming an account. Since I'm afraid of the related comment made by the previous user name and how the new name user can go back and delete the old name comment.

Link to Github repo: https://github.com/sorryma3er/NearbyExplorer.git

Link to demonstration vdo: https://youtu.be/G1WJXjf-Ev8