Enhancing the TrekBuddy - Travel Mobile Application - with React Native, Firebase Integration, and EAS Expo Deployment

Anh Nguyen

Haaga-Helia University of Applied Sciences
Bachelor of Business Information Technology
Product-based
2025

Abstract

Author(s)
Anh Nguyen

Degree
Bachelor of Business Information Technology

Report/Thesis Title
Enhancing the TrekBuddy Mobile Application with React Native, Firebase Integration, and EAS Expo Deployment

Number of pages and appendix pages
81 + 11

This thesis details the new feature development, implementation, and evaluation of TrekBuddy, a cross-platform mobile travel application designed to assist users in discovering, saving, and navigating to destinations with enhanced contextual and interactive features. The primary objective was to extend the application with user-centric tools that improve the travel planning experience through multilingual support, real-time data integration, and gamified engagement.

The application was developed using React Native, Firebase Firestore, and the Google Places API, with deployment handled via Expo Application Services (EAS). Key features include recommended restaurant listings based on user-selected cities, real-time weather display, language switching between English and Vietnamese, daily login streak, and a quiz-based gamification system that rewards users through points and streaks. Development followed an iterative process inspired by Agile principles, beginning with wireframes and prototypes, and advancing through continuous testing and feature refinement.

Evaluation through a user survey (N=6) indicated high usability (83.3% rated navigation as easy or very easy) and strong engagement with the new features—particularly the language toggle, weather display, and quiz game, all of which received 100% positive responses. Feedback also revealed areas for future enhancement, including clearer onboarding, improved restaurant detail, and the addition of a user-generated review system. Testing limitations were noted due to reliance on Expo Go, which restricted automated testing of advanced interactions.

This project demonstrates the feasibility of integrating multilingual support, location-based services, and gamification into a single, lightweight travel app. While results are promising, the evaluation acknowledges limitations in testing scope and sample size, suggesting further development and user research to refine TrekBuddy's potential in the mobile travel app landscape.

Key words
TrekBuddy, mobile application, travel app, React Native, Firebase, Firestore, Google Places API, cross-platform development, EAS Expo deployment, multilingual support, real-time weather, restaurant recommendations, quiz game

Table of contents

1 Introduction

In recent years, mobile applications have transformed the way people plan and experience travel, driven by the widespread adoption of smartphones and the demand for all-in-one digital solutions. This thesis presents the development of TrekBuddy, a mobile application designed to unify the fragmented travel experience by offering a single platform for discovering destinations, navigating with maps, and saving favorite locations.

Beyond its previously developed core functionality, TrekBuddy now includes real-time weather integration, multilingual support, a gamified quiz system to earn points and credits, daily login streak tracking and personalized restaurant recommendations—enhancing user engagement and delivering a more personalized travel experience. These newly developed features reflect a deeper focus on interactivity, localization, and user-centered design.

Developed independently, TrekBuddy posed unique challenges in architecture, feature planning, and technology integration. This thesis explores both the technical implementation and the self-managed development process, offering practical insights into building a scalable and feature-rich mobile app that aligns with the evolving expectations of today's travelers.

1.1 Background of the TrekBuddy Project

Travel planning has shifted from traditional methods to mobile-first experiences, driven by the rise of smartphones and global tourism. Despite the variety of travel apps available, many fall short by offering fragmented solutions—requiring users to switch between separate apps for discovery, saving, and navigation. Others suffer from generic recommendations and complex interfaces.

With advancements in cross-platform development tools like React Native and backend services like Firebase, developers can now build efficient, scalable apps with streamlined authentication, real-time data, and responsive UIs. TrekBuddy was developed to address these gaps by offering an all-in-one platform that combines destination discovery, real-time weather, multilingual support, personalized recommendations, and gamified user engagement—providing a modern, user-friendly travel companion for today's travelers.

1.2 Objectives and Scope

The primary objective of this thesis is to extend and enhance an existing mobile travel application, TrekBuddy, by designing and implementing new features that improve the overall user experience and functionality. The application aims to assist users in planning their trips by enabling them to view real-time weather conditions, explore recommended restaurants, change multiple languages

between English and Vietnamese, daily streak tracking by login, and support user engagement with a quiz-based gamification system that allows users to earn points.

A second objective of this thesis is also to support my personal learning and growth in mobile development, particularly in applying and expanding knowledge of Firebase Firestore in a new development environment. Previously, I had a chance to collaborate with Swiss students to develop a mobile gaming application for children using Flutter and Dart, where Firebase was implemented to manage user data. While that project provided foundational experience with Firebase, this thesis builds upon that knowledge by exploring Firebase integration within a React Native ecosystem using JavaScript. Specifically, the project is designed to build a strong understanding of cross-platform application development using React Native, Firebase, and Expo Application Services (EAS). I also set a goal to learn best practices for state management, internationalization (i18n), and API integration, particularly focusing on how to design, implement, and test scalable mobile features independently. Throughout the project, I aims to deepen understanding of user authentication, real-time database management, and secure data handling, while also learning how to adapt Firebase solutions across different frameworks, and this project introduced me to Expo and Expo Application Services (EAS), which were entirely new tools at the start of development. These learning outcomes are reflected on and critically assessed in the final evaluation chapter of the thesis.

To ensure the project outcomes are verifiable, the success of each implemented feature is evaluated through functional testing, usability reviews, and system compatibility checks. The technical feasibility and performance of each integration are also discussed.

The scope of this thesis encompasses the end-to-end development of the TrekBuddy mobile application using React Native, with a focus on creating a user-friendly, cross-platform travel assistant. Core tasks included building the front-end interface, integrating real-time weather updates via the OpenWeatherMap API, enabling secure data handling and user authentication through Firebase services, and incorporating location-based features using the Google Places API and React Native Maps. Additional functionalities such as multilingual support, a gamified quiz system, restaurant recommendations, and daily streak tracking were implemented to enhance user engagement and accessibility. The application was deployed and tested using Expo Application Services (EAS), with APK builds generated locally for testing on Android devices. The project also involved comprehensive documentation and evaluation of each feature, assessing both technical performance and user experience to inform future improvements and ensure scalability.

1.3 Research Questions and Development Tasks

To achieve these objectives, the thesis will address the following key research questions:

- How can integrating real-time weather data, restaurant recommendations, multilingual support, gamification, and daily login streak improve the overall user experience in a travel planning mobile application?
- What are the primary technical challenges in integrating new features within a React Native application using Expo Application Services (EAS)?
- How effective is internationalization (i18n) in creating a scalable and user-friendly cross-platform mobile application?
- How do users perceive the usability and effectiveness of the newly implemented features, and what insights can be drawn from user testing to guide future improvements?

The research questions were formulated based on addressing the following underlying concerns:

- What gaps in functionality or user engagement exist in current travel planning applications?
- How can internationalization and API integrations be optimized for a smooth user experience in multilingual, cross-platform mobile apps?
- What tools, design patterns, and workflows best support a student developer learning full-stack mobile development independently?

The development tasks include:

- Designing and implementing a real-time weather display feature using external weather APIs.
- Building a restaurant recommendation feature using the Google Places API with city-based filtering and image support.
- Enabling multilingual support with a language toggle system between English and Vietnamese using i18n.
- Developing a quiz-based gamification system that allows users to earn and store points or credits in Firestore.
- Integrating Firebase Firestore for real-time data storage, retrieval, and updates across multiple features.
- Conducting usability testing and analyzing user feedback to refine app features and interactions.
- Reflecting on the learning process, challenges encountered, and skills gained throughout the development cycle.

The development tasks are directly tied to the research questions and thesis objectives, ensuring that:

- Each new feature corresponds to a specific research question (e.g., the quiz system addressing user engagement through gamification).

- Technical hurdles are addressed through iterative development, testing, and documentation.
- The development process not only enhances the application but also supports personal growth in mobile development through applied learning and evaluation.

1.4 Structure of the Thesis

This report is structured to provide a comprehensive overview of TrekBuddy's development process. It begins with an Introduction, outlining the project's background, objectives, and significance. The Application Infrastructure section discusses the technologies used for new features development—such as the React Native framework, Firebase Authentication with Firestore, and Google Places API—and the rationale behind their selection. The Application Implementation Process details the design and prototyping phase, existing and newly developed features, as well as the testing strategies and deployment preparation. A dedicated section on Deployment Using EAS highlights the local build process, app signing, and configuration steps. The Discussion section evaluates the app’s core functionalities, user experience, and identifies strengths, weaknesses, and potential areas for improvement. Finally, the Conclusion summarizes key findings, contributions of the thesis, and offers recommendations for future work.

2 Theoretical Framework

This section outlines the theoretical foundation underlying the new feature development in the TrekBuddy mobile application. The core objective is to enhance user experience and application scalability by integrating real-time weather data, restaurant recommendations, multilingual support (internationalization), and gamification. These enhancements are grounded in established theories and practices in mobile user experience design, information systems, and cross-platform development.

2.1 Enhancing User Experience in TrekBuddy

User experience (UX) in mobile applications is influenced by usability, functionality, contextual relevance, and emotional engagement (Hassenzahl, 2010). In the context of travel planning, users value real-time, location-specific information that helps them make decisions on the go. The integration of real-time weather data directly supports situational awareness, which is critical in dynamic travel environments (Endsley, 1995). Knowing whether it will rain or shine can influence a traveler's choice of destination, activity, or attire.

Restaurant recommendation systems, often powered by APIs like Google Places, fall under the domain of context-aware services (Adomavicius & Tuzhilin, 2011). These systems improve decision-making by personalizing suggestions based on user location or preferences, thereby increasing satisfaction and perceived usefulness. Such context-based content delivery enhances the perceived quality and efficiency of the travel planning experience.

Gamification—the use of game mechanics in non-game contexts—has been shown to improve user motivation, engagement, and retention (Deterding et al., 2011). In the TrekBuddy application, the quiz-based reward system adds an element of challenge and achievement that encourages users to interact with the app more frequently. This aligns with self-determination theory (Deci & Ryan, 2000), which suggests that competence and autonomy are key drivers of intrinsic motivation.

The daily login streak feature in TrekBuddy is grounded in gamification and motivation theory, aiming to increase user engagement through habit formation. By visually rewarding users for consecutive daily use (e.g., a 🔥3-day streak), it leverages positive reinforcement and the desire to avoid breaking a streak, which are known to encourage repeated behaviors (Skinner, 1953; Deci & Ryan, 2000). This simple mechanic fosters a sense of progress and accomplishment, aligning with self-determination theory and enhancing user retention by making engagement feel rewarding and continuous.

Together, these features aim to improve not only the functionality of the application but also the overall hedonic and pragmatic qualities of the user experience (Hassenzahl, 2005).

2.2 Internationalization (i18n) and Cross-Platform Scalability

As mobile apps target a global audience, internationalization (i18n) becomes essential in ensuring usability across different languages and cultural contexts. Internationalization refers to the process of designing an application in a way that it can be easily adapted to various languages without requiring engineering changes (Schwartz et al., 1999). This is especially relevant in bilingual or multilingual regions where user preference for language greatly impacts perceived accessibility and satisfaction.

Internationalization not only improves user inclusivity but also contributes to scalability by enabling the app to be deployed in multiple markets with minimal overhead. When paired with cross-platform development frameworks like React Native, i18n further simplifies the process of maintaining a consistent and localized experience across Android and iOS platforms (Singh & Wesson, 2009). This is critical in resource-constrained student-led development projects, where maintaining a single codebase increases efficiency.

From a theoretical perspective, localization (the adaptation of content to fit specific cultural norms) is linked to the concept of cultural usability (Marcus & Gould, 2000). Applications that fail to support native languages often experience lower engagement, highlighting the importance of linguistically and culturally sensitive design.

3 Application Infrastructure

TrekBuddy is built on a well-structured application infrastructure to provide a seamless, scalable, and efficient user experience. Given the necessity for cross-platform interoperability, real-time data management, and reliable location-based services, the project combines modern front-end and back-end technology. React Native is the primary framework for creating a high-performance mobile application that runs on both iOS and Android from a single codebase. Firebase offers a cloud-based backend solution for authentication and data storage, making it possible to manage users and databases efficiently without the need for a dedicated server. Furthermore, Google Places API and Google Maps API provide sophisticated location-based services like place searches, navigation aids, and interactive mapping.

Figure 1. Tech Stack architecture

3.1 React Native framework

React Native (RN) is a widely adopted mobile application framework that utilizes JavaScript to develop natively-rendered applications for both iOS and Android platforms using a single codebase. Initially introduced by Facebook as an open-source project in 2015, React Native has rapidly gained popularity and has become one of the leading solutions for mobile application development. Its efficiency and versatility have led to its adoption by several major applications, including Instagram, Facebook, and Skype, demonstrating its capability to power large-scale mobile platforms (GeeksforGeeks, March 2024).

3.1.1 Benefits from using React Native

Bhagavatiprasad Vaghela (February 2023) indicated that React Native has become a highly successful framework in modern mobile application development due to its efficiency, flexibility, and cost-effectiveness. One of its key advantages is live and hot reloading, which allows developers to see code changes instantly without recompilation, significantly enhancing productivity. Additionally, React Native follows a "write once, run anywhere" approach, enabling developers to use a single codebase for both iOS and Android applications with minimal platform-specific modifications, making it a cost-effective and time-saving solution. Built on JavaScript, one

of the world's most widely used programming languages, React Native benefits from a large developer community and extensive support. Furthermore, it ensures a native look and feel by leveraging platform-specific components that render natively on each device, providing a seamless and high-performance user experience.

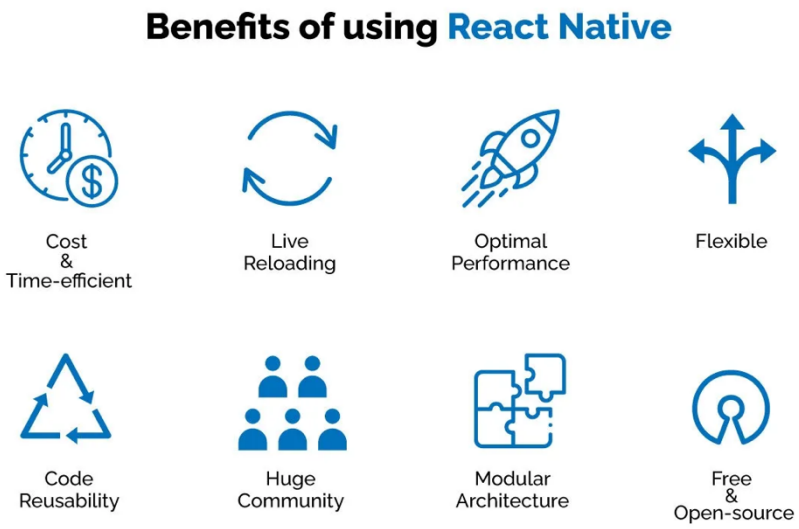


Figure 2. React Native benefits (Invozone Blog, Sadia Aziz)

3.1.2 Why React Native for TrekBuddy project?

Several aspects inspired the choice of React Native for TrekBuddy development, most notably its cross-platform interoperability, which allows a single codebase to run efficiently on both iOS and Android (React Native, n.d.). Its rapid development cycle, aided by hot reloading, enables developers to observe real-time changes without recompiling the entire software, considerably increasing productivity (Eisenman, 2021). Furthermore, React Native features a large developer community, abundant documentation, and a diverse set of third-party libraries, making it easy to troubleshoot and expand functionality (Vladimir, 2017). Additionally, React Native enhances performance by utilizing native components, resulting in a seamless user experience while balancing efficiency and responsiveness (React Native, n.d.).

To enhance TrekBuddy's functionality, several core and third-party components are utilized. The View and Text components serve as fundamental building blocks for structuring the UI, while React Navigation ensures smooth transitions between screens (React Navigation, n.d.). Furthermore, Gesture Handler and Animated API enable interactive animations for an engaging user experience (Software Mansion, n.d.). Expo Modules simplify access to device-specific functionalities such as camera, push notifications, and location tracking, reducing the complexity of native module

integration (Expo, n.d.). React Native also integrates seamlessly with Firebase, which provides authentication and real-time database capabilities (Google Firebase, n.d.), as well as Google Places API and Google Maps API, which enhance the app's location-based services (Google Cloud, n.d.a, n.d.b). This cohesive integration of technologies allows TrekBuddy to deliver a robust, user-friendly, and high-performance travel application.

3.2 Firebase Authentication and Firestore

Geeksforgeeks (February 2025) informed that Firebase, developed by Google, is a robust platform that allows developers to efficiently build, manage, and scale applications. By providing a secure and reliable backend, Firebase eliminates the need for complex server-side programming, streamlining the development process. Its cross-platform compatibility extends to Android, iOS, web, and Unity, making it a flexible and widely adopted solution. With features such as real-time cloud storage, authentication, and NoSQL database support, Firebase facilitates smooth data management and synchronization. Its cloud-based infrastructure accelerates development, enhances security, and ensures effortless scalability, enabling developers to focus on delivering a seamless user experience.

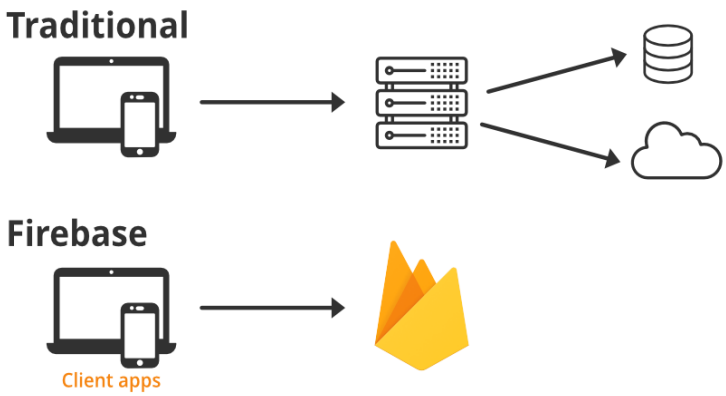


Figure 3. Introduction to Google Firebase (Cyber Yodha)

Figure 3 compares traditional backend architecture with Firebase's cloud-based model. Unlike traditional systems that rely on multiple servers, Firebase allows direct client access to its cloud services, simplifying development and enabling real-time sync, authentication, and scalable storage.

3.2.1 Key Backend Services in Firebase

Firebase offers a comprehensive suite of backend services designed to assist developers in building and managing applications more efficiently. These services provide seamless integration,

real-time capabilities, and scalable solutions, reducing the need for extensive server-side management (GeeksforGeeks, 2025).

In the TrekBuddy project, only Firebase Authentication and Cloud Firestore were utilized to manage user authentication and data storage efficiently. Firebase Authentication was implemented to enable secure user sign-in and account management, allowing users to log in using email and password authentication. Meanwhile, Cloud Firestore, a NoSQL document-based database, was used to store and manage user-generated content, such as saved destinations, travel itineraries, and preferences. These Firebase services provided a scalable, real-time, and serverless backend solution, eliminating the need for manual backend development while ensuring a seamless and secure user experience.

- **Authentication** – Firebase Authentication simplifies user authentication by providing UI libraries and SDKs that support multiple authentication methods, including email/password, social logins (Google, Facebook, etc.), and phone authentication. This service automates user management, reducing the development effort required to implement secure login functionalities (GeeksforGeeks, 2025).
- **Cloud Firestore** – A NoSQL document database that allows developers to store, sync, and query data globally. Unlike the Realtime Database, Firestore structures data into documents and collections using a key-value format, supporting complex data types such as strings, binary data, and JSON trees (GeeksforGeeks, 2025).

Figure 4. Realtime data synchronization using Cloud Firestore (DailyHostNews)

Figure 4 illustrates Firebase Cloud Firestore's real-time data sync and management. It shows multiple devices connected to a cloud database, enabling seamless updates across platforms. Firestore uses a NoSQL document model with advanced querying, offline support, and scalability—ideal for structured data, live updates, and multi-user collaboration.

3.2.2 Why Firebase Authentication and Firestore for TrekBuddy?

Firebase Authentication and Cloud Firestore were chosen for the TrekBuddy project's backend to ensure security, scalability, and efficiency. Firebase Authentication simplifies user administration by enabling secure sign-ins through email and password authentication without requiring complex backend functionality (Google Firebase, n.d.). This service provides robust security with OAuth 2.0 and token-based authentication, reducing the risk of unauthorized access and streamlining the authentication process for users (Stallings, 2017).

Meanwhile, Cloud Firestore was implemented as the primary NoSQL database to store and manage user-generated content such as saved destinations, travel itineraries, and preferences. Firestore offers real-time data synchronization, ensuring that updates made on one device are instantly reflected across all connected devices, and its offline support allows users to access their saved data even without an internet connection—an essential feature for travelers (Google Firebase, n.d.). By leveraging Firebase Authentication and Cloud Firestore, TrekBuddy eliminates the need for dedicated backend infrastructure, ensuring a cost-effective, scalable, and reliable system that enhances the overall user experience (Harrington, 2019).

3.2.3 Set up Firebase modules for the React Native

Setting up Firebase for a React Native project involves integrating Firebase services using the Firebase SDK and configuring the project to communicate with Firebase. The process begins by creating a Firebase project in the Firebase Console and registering the app for both iOS and Android (Google Firebase, n.d.). After setting up the project, the required Firebase dependencies, such as `@react-native-firebase/app` and specific modules like `@react-native-firebase/auth` for authentication or `@react-native-firebase/firestore` for database management, are installed using `npm` or `yarn` (React Native Firebase, n.d.). Next, the Firebase Google Services configuration files (`google-services.json` for Android and GoogleService-Info.plist for iOS) must be added to the appropriate directories within the React Native project (Google Firebase, n.d.). For Expo-managed projects, Firebase setup is streamlined using third-party packages like `expo-firebase-auth` and `expo-firebase-core` (Expo, n.d.). Finally, Firebase is initialized within the app code, and authentication or Firestore services can be integrated using the Firebase SDK (React Native Firebase, n.d.). This setup ensures that the React Native application can securely interact with Firebase services, enabling authentication, real-time database management, and cloud functions within the app (Harrington, 2019).

```
{
  "expo": {
    "android": {
      "googleServicesFile": "./google-services.json",
      "package": "com.mycorp.myapp"
    },
    "ios": {
      "googleServicesFile": "./GoogleService-Info.plist",
      "bundleIdentifier": "com.mycorp.myapp"
    },
    "plugins": [
      "@react-native-firebase/app",
      "@react-native-firebase/auth",
      "@react-native-firebase/crashlytics",
      [
        "expo-build-properties",
        {
          "ios": {
            "useFrameworks": "static"
          }
        }
      ]
    ]
  }
}
```

Figure 5. An example `app.json` to enable the React Native Firebase modules App, Auth and Crashlytics

Figure 5 shows the Firebase configuration for both Android and iOS platforms, specifying the service account files and setting the application ID (e.g., `com.mycorp.myapp`) as an example.

For this TrekBuddy project, Firebase were installed by using following commands:

```
npm install firebase
```

Listing 2. Install Firebase package with the command

```
npm install @react-native-firebase/app
```

Listing 3. React Native Firebase's base app module

```
npm install @react-native-firebase/auth @react-native-firebase/firestore
```

Listing 4. Install Firebase Authentication and Firestore for project.

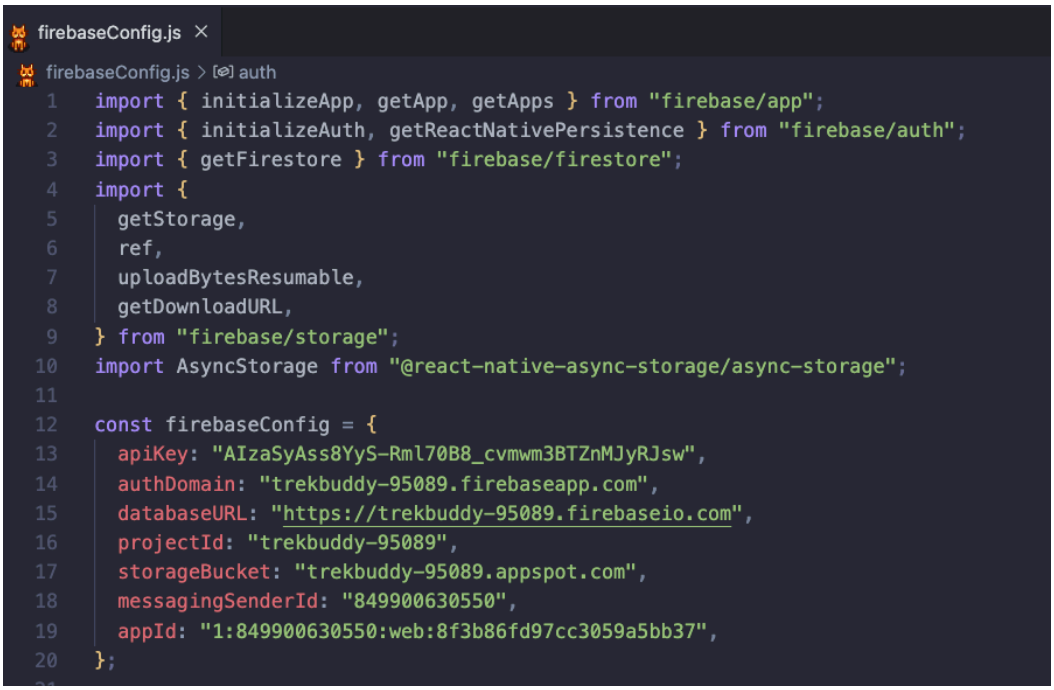


Figure 6. Firebase configuration file (`firebaseConfig.js`) for TrekBuddy.

Figure 6 shows the Firebase configuration file (`firebaseConfig.js`) used in the TrekBuddy project to initialize and integrate Firebase services.

3.3 Google Places API

The Google Places API is a web service that enables applications to access detailed place information, including geographical coordinates, business establishments, landmarks, and points of interest. It allows developers to integrate place search, real-time details, and user-generated content to enhance location-based experiences. By incorporating this API, applications can retrieve addresses, contact details, ratings, reviews, photos, and operating hours for various places. The API processes HTTP requests to return structured geographic data and imagery, making it a valuable tool for delivering accurate and dynamic location information (Google Cloud, n.d.a).

3.3.1 Advantages of using Google Places API

The Google Places API empowers developers to build location-aware features that provide users with reliable and detailed geographic data. It is based on a comprehensive and regularly updated place model. Example use cases include displaying property listings in targeted areas, showing place details for delivery or pickup services, and listing parks with user-submitted content. The API also supports travel planning by offering contact information, reviews, and pricing details for establishments along specific routes.

Applications can retrieve place search results based on various user queries, including text input, nearby searches, and category-based queries. Furthermore, developers can access detailed place attributes such as operating hours, summaries, user reviews, and high-resolution photos. The API includes autocomplete functionality to refine user input and improve the overall search experience. Additionally, the use of place IDs—unique identifiers for locations in Google Maps and Places databases—enables seamless retrieval of extended details. These IDs can be obtained via the Geocoding API, Routes API, or Address Validation API (Google Cloud, n.d.b; Google Cloud, n.d.c; Google Cloud, n.d.d), and used within the Places API to enhance user interactions by delivering full addresses, contact data, and user ratings (Google Cloud, n.d.a).

3.3.2 Why Google Places API for TrekBuddy?

In the TrekBuddy project, Google Places API is used to provide real-time destination search, place details, and personalized recommendations for travelers. By integrating this API, the app allows users to search for nearby attractions, retrieve accurate place information, and get location-based recommendations based on their travel preferences (Google Cloud, n.d.a). This enhances the overall user experience by making it easier for travelers to discover and explore new destinations efficiently (Shneiderman & Plaisant, 2010).

3.3.3 Install Google Places API in React Native project

To integrate the Google Places API into the TrekBuddy application, a Google Cloud project must first be created, and the API must be enabled via the Google Cloud Console. After generating an API key, dependencies such as `react-native-google-places-autocomplete` are installed. For Expo projects, only this package is needed, while for bare React Native projects, `react-native-maps` is also required (React Native Google Places Autocomplete, n.d.).

Next, the API key is embedded into the project and configured within the Google Places Autocomplete component, which supports location search and real-time detail retrieval. For Android, the key is added to the `AndroidManifest.xml` file, and for iOS, it is configured in the `AppDelegate.m` file. After completing these steps and rebuilding the app, users can enjoy seamless integration of place search, enhancing their navigation and exploration experience within TrekBuddy (Google Cloud, n.d.a; React Native Google Places Autocomplete, n.d.).

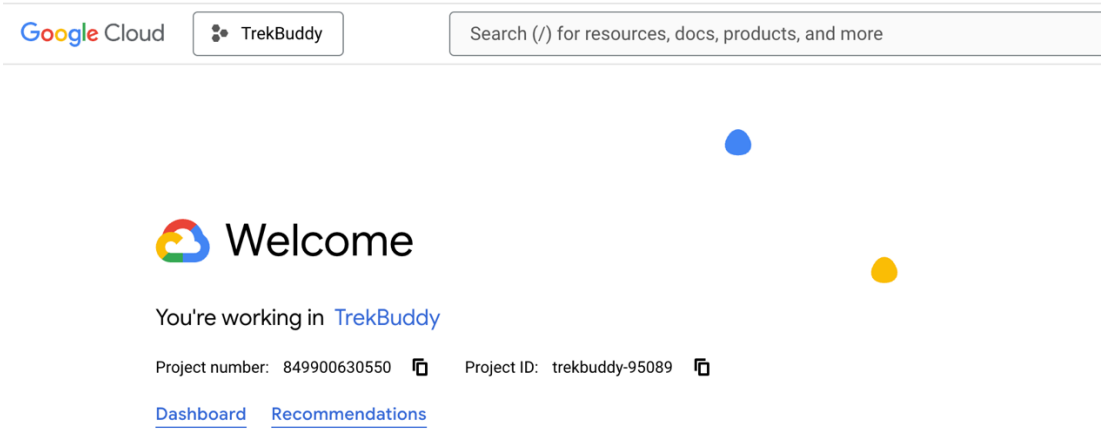


Figure 7. Google Cloud Console window for TrekBuddy project

After creating “TrekBuddy” project in Google Cloud Console, Google Places API was installed in this project by the following commands:

```
expo install react-native-google-places-autocomplete
```

Listing 5. Install Google Places required package for Expo-managed workflow

```
npm install react-native-google-places-autocomplete react-native-maps
```

Listing 6. Install Google Places required package for Bare React Native workflow

Then, create a `.env` file in the root to store `GOOLE_API_KEY`, which is obtained from Google Cloud Console.

```
const fetchFamousCities = async () => {
  if (searchTerm.trim() === "") return;

  try {
    const response = await fetch(
      `https://maps.googleapis.com/maps/api/place/textsearch/json?query=cities+in+${searchTerm}&key=${GOOGLE_API_KEY}`
    );
```

Figure 8. Fetching famous cities using Google Places API in TrekBuddy.

Figure 8 illustrates a snippet defines an asynchronous function (`fetchFamousCities`) that uses the Google Places API Text Search endpoint to retrieve a list of famous cities based on a user's search input, helping TrekBuddy provide relevant travel destination suggestions.

4 Application Implementation Process

4.1 Existing Features in TrekBuddy

The TrekBuddy mobile application is a location-based travel companion that was developed using **React Native** with **Expo**, integrated with a **Firebase** backend and **Google Place APIs** to offer real-time data, intuitive UI/UX, and robust user authentication. This section describes the core features of the TrekBuddy app, covering both front-end and back-end implementations, the technologies involved, and what users can do within the application.

4.1.1 Front-End

Design and Prototyping

The goal of TrekBuddy's design and development phase was to create an easy-to-use interface that would improve the trip planning process. The team used Figma to create wireframes, prototypes, and user interface elements in an iterative, user-centered process that was informed by feedback and journey mapping. Important functions such as itinerary building, destination search, and interactive maps were improved to guarantee accessibility and smooth cross-device navigation. In order to reflect its captivating brand identity, the app features a dual-theme design, with light mode including gentle pastel pink tones and dark mode featuring a sleek gray background. It also features playful, rounded font and a bold, amiable logo.

Figure 9. TrekBuddy's logo

Navigation System

TrekBuddy employs a structured navigation system, enabling smooth transitions between screens. The app uses a **Stack Navigator** for hierarchical screen flows (e.g., Login → Home) and a **Bottom Tab Navigator** for core screens such as Explore, Saved, Settings, and Profile.

TrekBuddy uses a combination of Stack Navigation and Bottom Tab Navigation from React Navigation to allow smooth transitions between screens like Login, Home, Settings, Profile, and Saved Places.

On this screen, users are able to navigate between core sections such as Explore, Saved, Settings, and Profile; transition through screens such as sign-up, login, and policy pages; and view tab icons and active/inactive color states for enhanced navigation clarity.

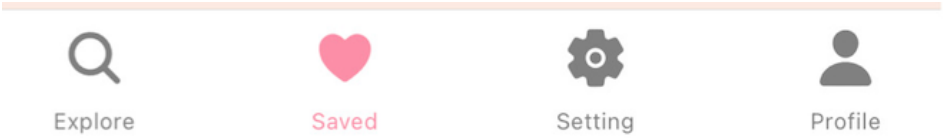


Figure 10. TrekBuddy's bottom tab navigation screen.

User Authentication

TrekBuddy supports secure user registration and login using email and password. The authentication process includes input validation, email verification, and error handling. Once signed in, user data is stored in Firestore for personalized experiences.

TrekBuddy used Firebase Authentication and AsyncStorage to integrate secure user authentication via email and password, with persistent login and email verification handled through Firebase.

On this screen, users can register a new account with email and password; receive verification email before accessing full features; log in securely and remain signed in; handle login errors with clear feedback messages; and sign out or handle login errors effectively.

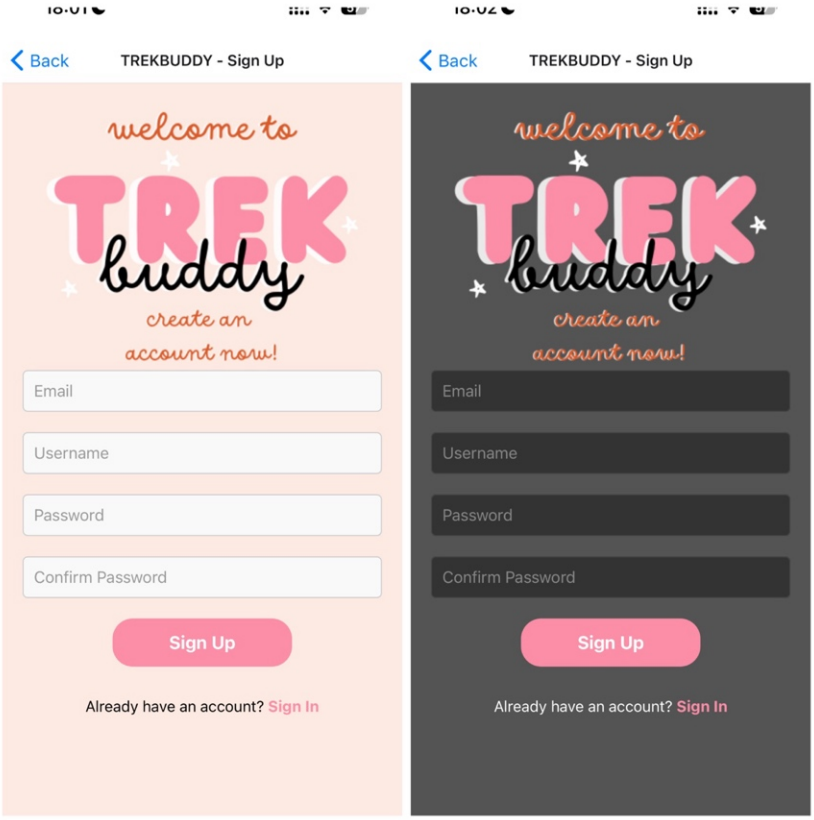


Figure 11. TrekBuddy's sign up screen.

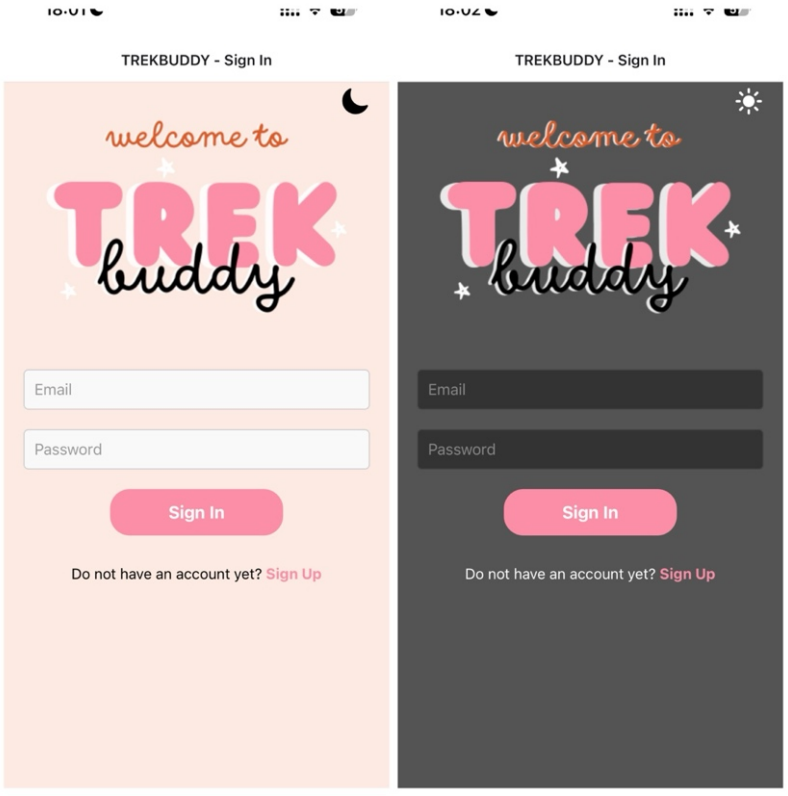


Figure 12. TrekBuddy's sign in screen.

Theming and UI Customization

TrekBuddy supports dynamic theming, allowing users to toggle between light and dark modes directly from the Settings screen. Theming is handled globally using the Context API and includes accent color customization (e.g., #fc8fa7 for selected tabs).

TrekBuddy used React Context API, StyleSheet, Ionicons to integrate the switching light and dark mode theme by toggle.

On this screen, user can toggle themes on the Settings screen; view accent color highlights (e.g., pink for active tabs); and experience consistent UI across all screens.

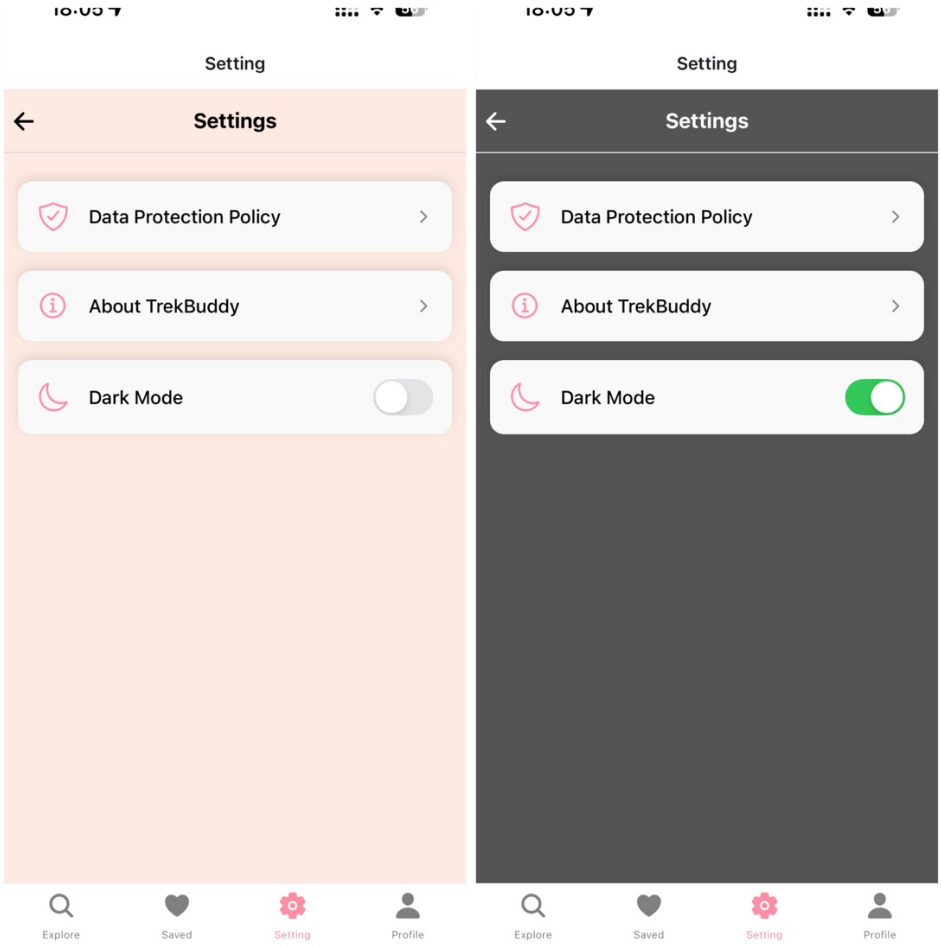


Figure 13. TrekBuddy's settings screen.

Home Screen Image Slider

The Home screen features a dynamic image slider that showcases scenic travel destinations. The slider supports animated transitions, auto-scrolling, and responsive effects to enhance user engagement.

This feature is implemented by using React Native Reanimated, ScrollView, FlatList, and custom animated components. Further details about the library and its capabilities can be found in the official documentation (Software Mansion, 2025).

On the Homepage, user can view a rotating carousel of travel locations; enjoy smooth animations that highlight popular spots; and experience visual inspiration to explore new places.

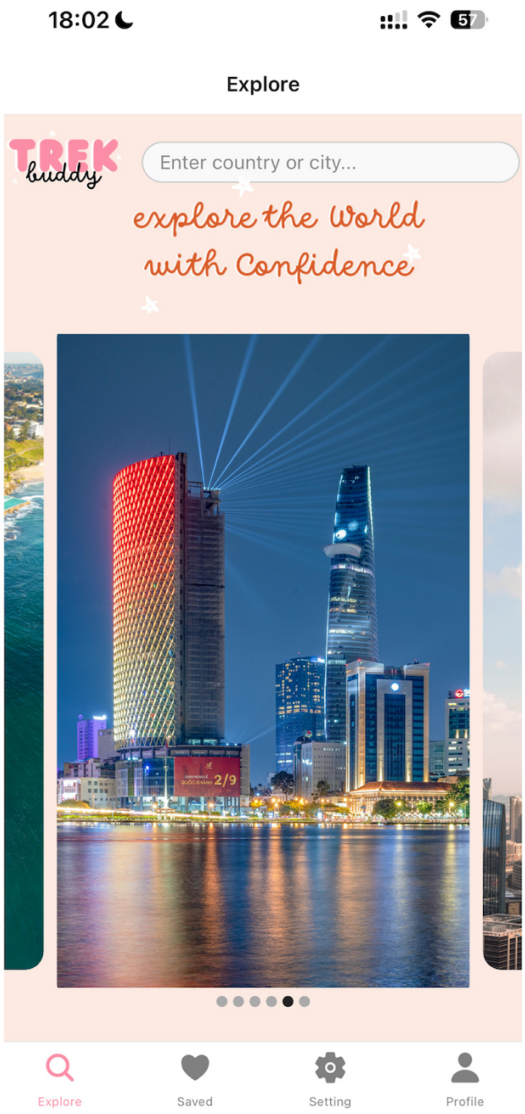


Figure 14. TrekBuddy's slider images on Home screen.

Maps Integration

Users can view interactive maps with support for location markers. The map centers on a predefined location or a user-selected destination and allows zooming and interaction.

TrekBuddy used `react-native-maps` and Apple Map as a default when using Expo to display the selected destination on Map.

On this screen, user are able to view map centered on a specific city or destinations; interact with map by zooming in or out; navigate through a visual map interface.

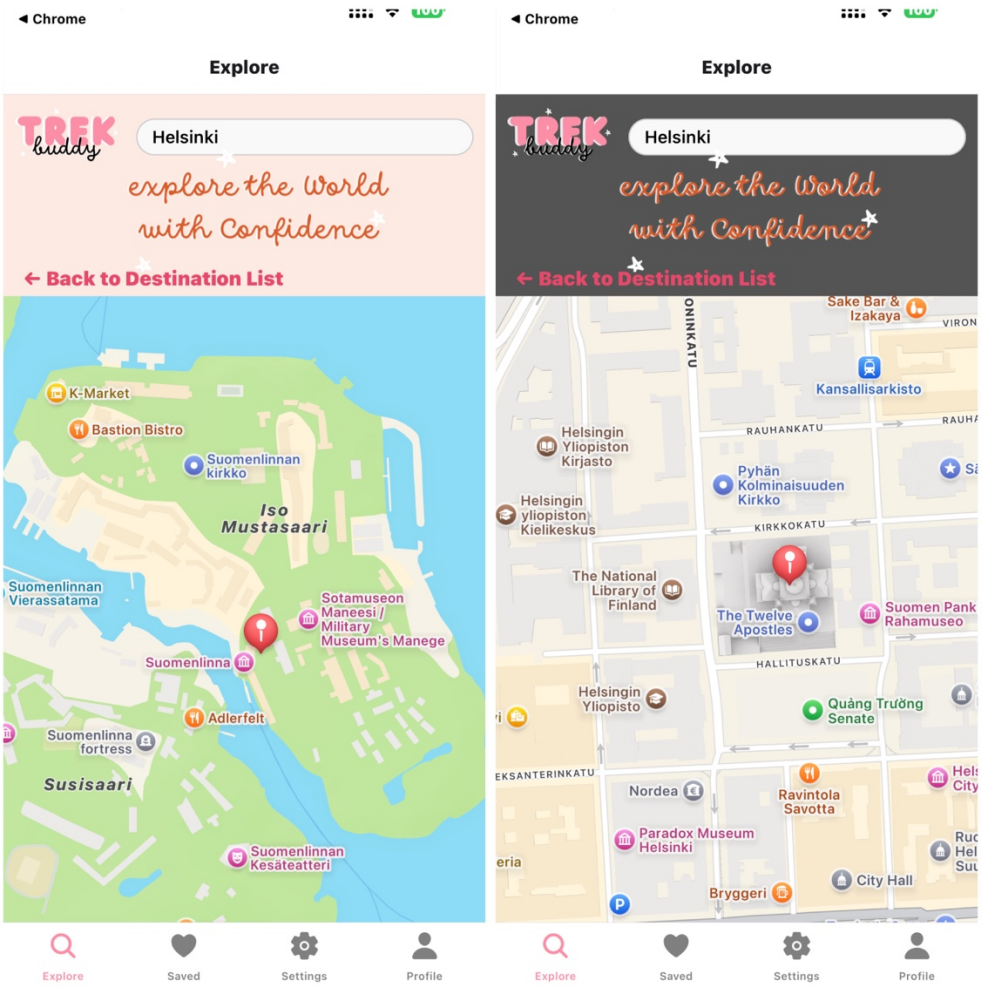


Figure 15. Map screen displays destination marker using Apple Map.

Google Places Search

TrekBuddy allows users to search for cities and tourist destinations using the Google Places API. Based on user input, the app fetches city details and attractions, displaying them as cards with names, photos, and addresses.

This feature uses Google Places API to fetch the countries, cities, and destinations.

With this feature, user can search for cities or countries; explore tourist attractions in selected locations; switch between city view and destination view; and view real-time search results with images and details.

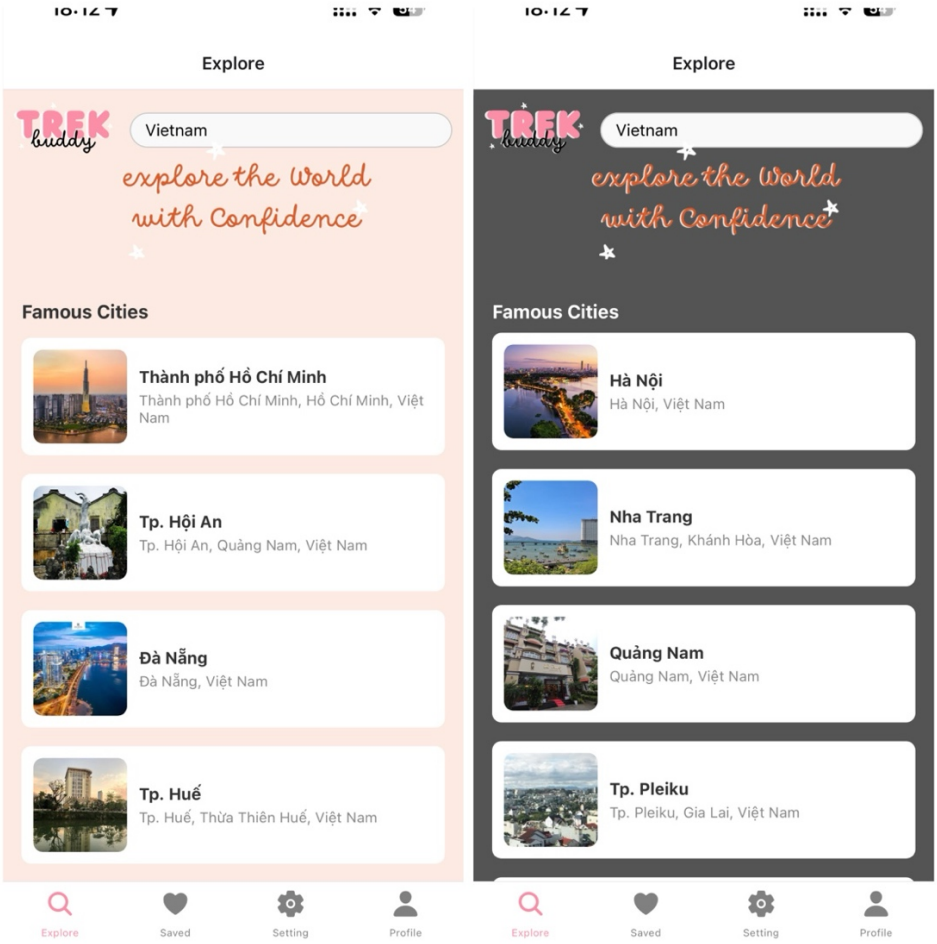


Figure 16. TrekBuddy's search cities or countries screen.

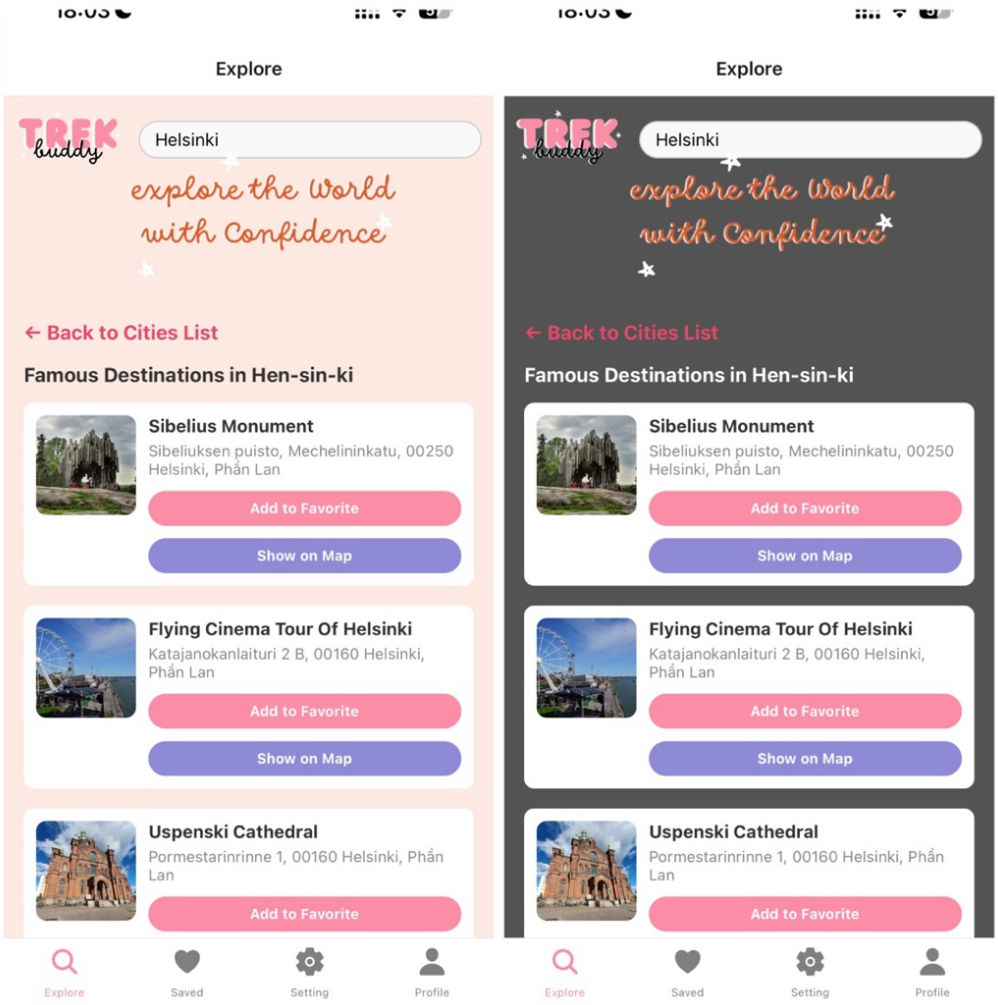


Figure 17. TrekBuddy's search destinations screen.

Profile and User Data Management

The profile screen allows users to view and edit personal information. This includes the ability to update their display name and change profile pictures, which are uploaded to Firebase Storage and linked in Firestore.

TrekBuddy uses Firebase Firestore, Firebase Storage, and expo-image-picker to help users manage their profile.

On this screen, user can view current username and profile picture; update username and reflect changes instantly; upload a new profile image from device gallery; store and retrieve profile info securely from Firebase; and sign out from the profile screen.

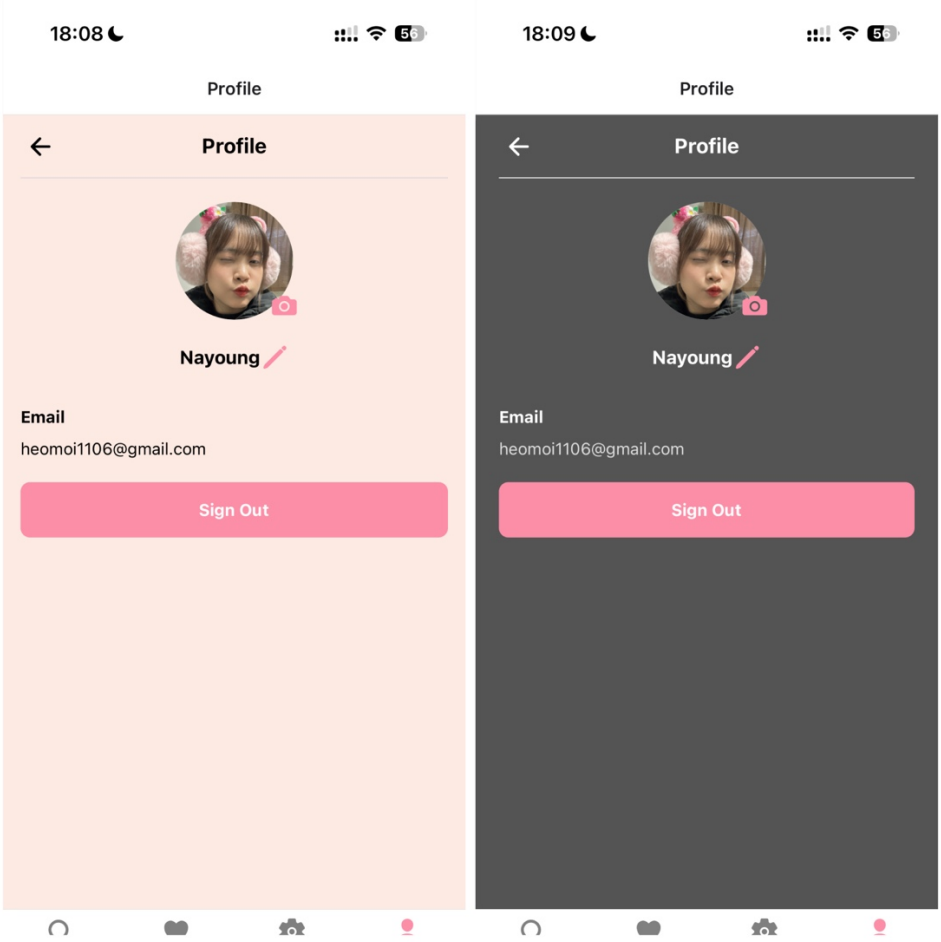


Figure 18. TrekBuddy's user profile screen.

Saved Screen for Collection Management

This screen allows users to view, create, delete, and navigate their saved collections. Firestore handles back-end storage and retrieval, while the UI dynamically adjusts based on user selection.

This feature uses Firebase Firestore, FlatList, TouchableOpacity, Modal, Alert and TextInput to implement the save functionality.

With this feature, user can fetch saved collections and view them; add a new collection with unique name validation; delete collections or remove individual saved places and use FlatList for smooth rendering of saved content; save places to collections; receive alerts upon successful save; remove individual items or entire collections, and view saved destinations in categorized lists.

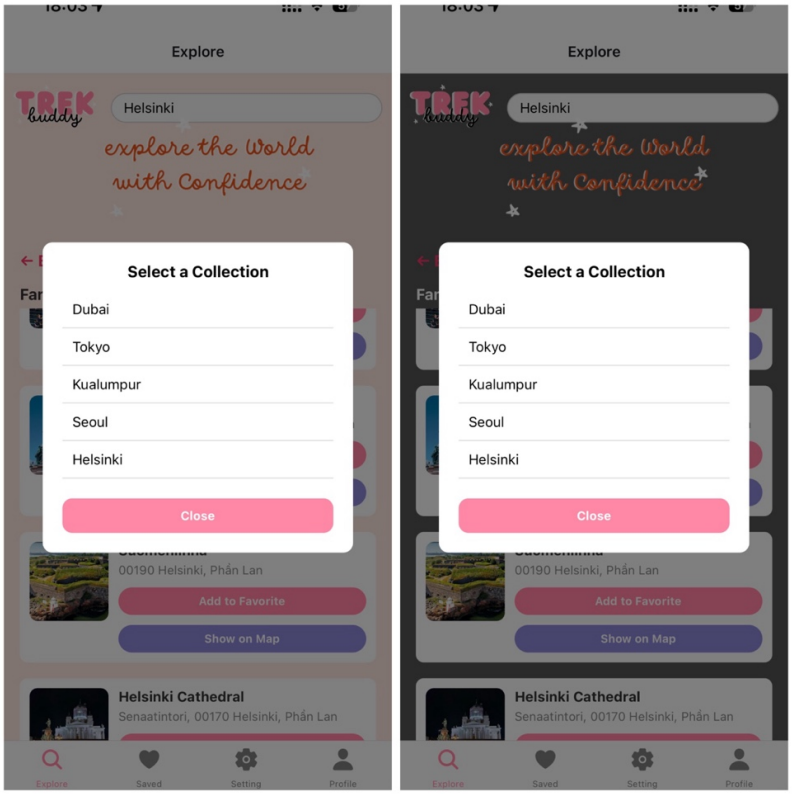


Figure 19. TrekBuddy's save destinations to collection screen.

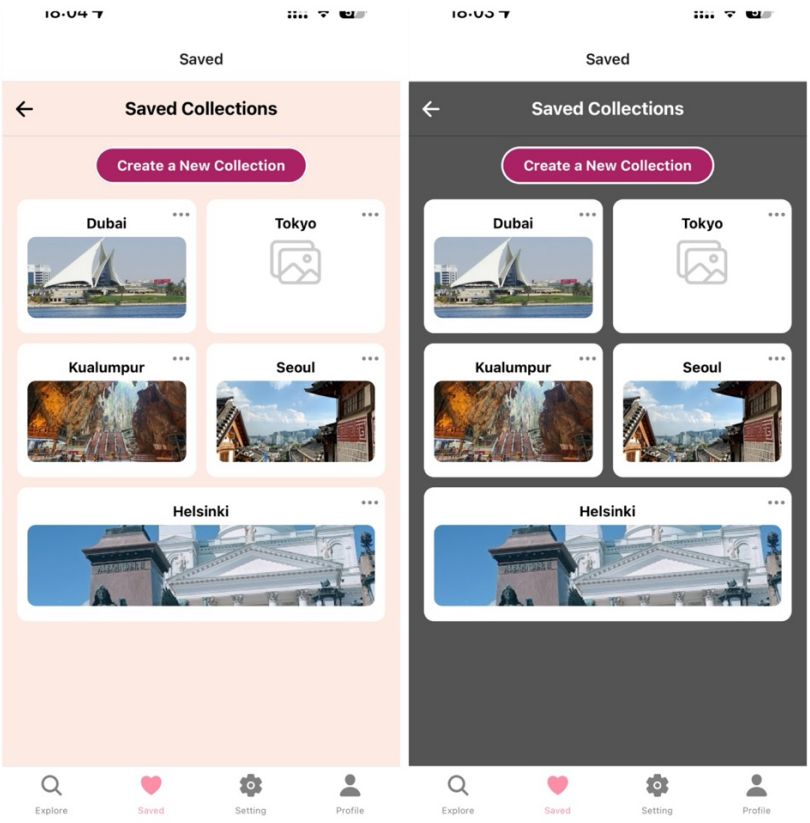


Figure 20. TrekBuddy's saved collections screen.

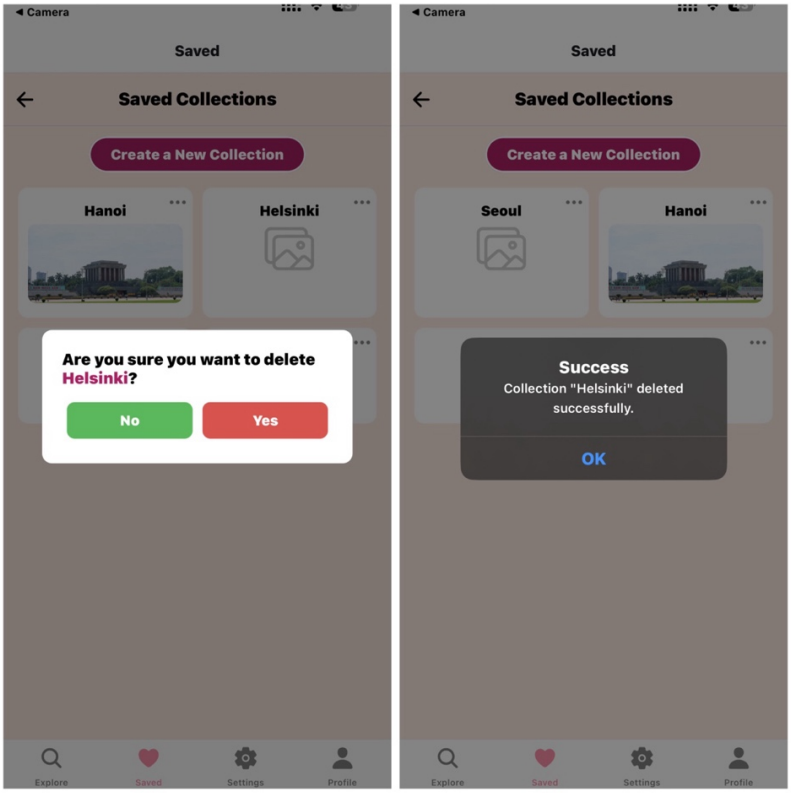


Figure 21. TrekBuddy's delete collection and success alert.

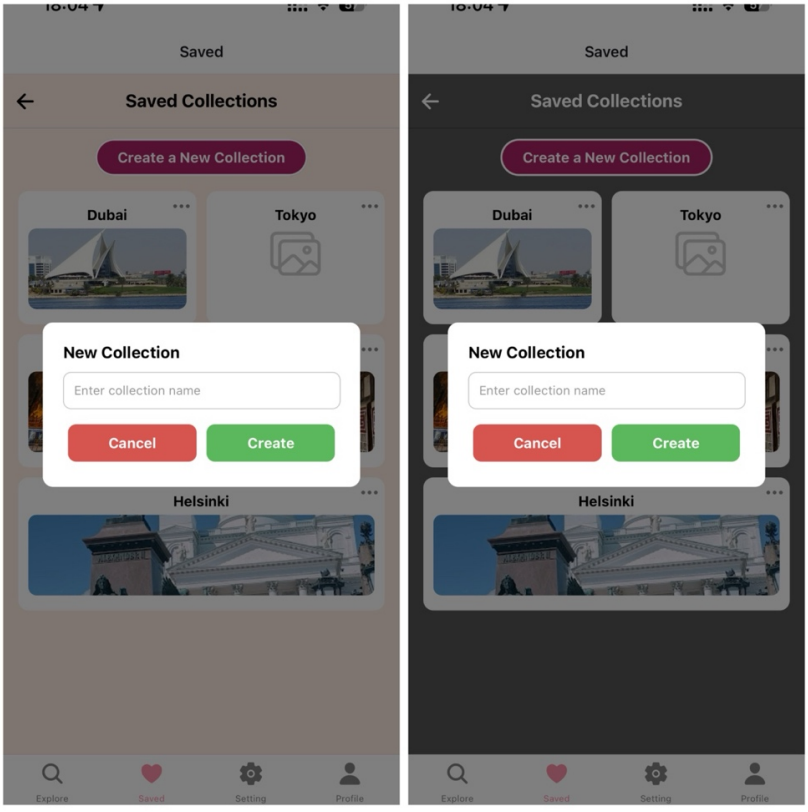


Figure 22. TrekBuddy's create collection screen.

4.1.2 Back-End

Firebase Backend Architecture

TrekBuddy uses Firebase (Firestore, Auth, Storage, Cloud Functions)

TrekBuddy leverages Firebase to manage back-end functionality without dedicated server maintenance. Firebase services are used for authentication, data storage, image uploads, and serverless logic execution.

- Firestore: NoSQL real-time database.
- Firebase Auth: User authentication with token-based security.
- Firebase Storage: For storing profile images.
- Cloud Functions: Executing backend tasks (notifications, data processing).

Firestore Database Design

a) NoSQL Structure

Firestore stores data using a document-collection model. Each user has a unique document inside the Users collection, which includes a nested userCollections subcollection for saved trips.

b) Database Schema

Users (`/Users/{userId}`): Stores user information including authentication details, profile information, and user preferences.

User Collections (`/Users/{userId}/userCollections/{collectionId}`): Stores collections of saved destinations for each user.

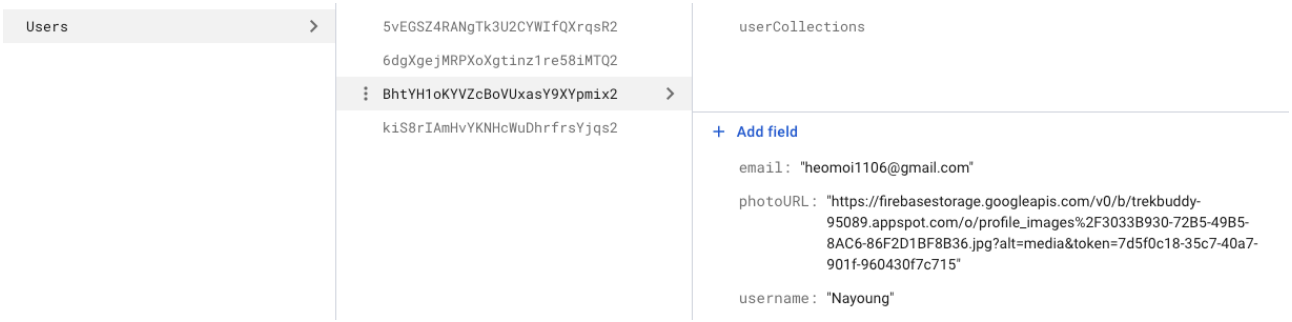


Figure 23. Firestore User Database Structure in TrekBuddy

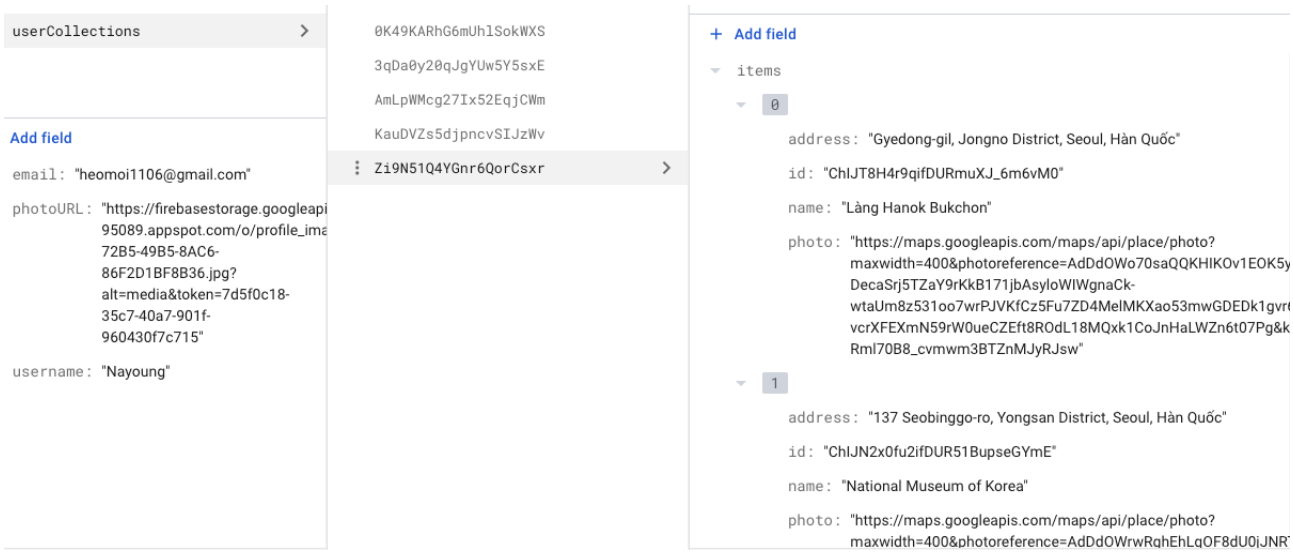


Figure 24. User Collections Data Structure in TrekBuddy Firestore Database.

Firebase Authentication and Security

a) User Auth

Users are authenticated via email/password. Upon sign-in, a secure token is generated, and user data becomes accessible in the app.

User can sign up and verify their email; log in securely with persistent sessions; and only access their own data (via Firestore rules).

b) Firestore Security Rules

Security rules are applied to ensure users can only read/write their own data.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    // Rule for the Users collection
    match /Users/{userId} {

      // Allow authenticated users to read and write only their own documents
      allow read, write: if request.auth != null && request.auth.uid == userId;

      // Nested rule for collections under each user
      match /{subCollection=**} {
        allow read, write: if request.auth != null && request.auth.uid == userId;
      }
    }
  }
}
```

Figure 25. Firestore Security Rules in TrekBuddy

4.2 Application requirements

In the TrekBuddy application, users engage with features that enhance their travel experience through a single-user interface. Built with Firebase Authentication, users can securely create accounts, log in, and manage their profiles. The app integrates Google Places and Maps APIs to allow destination search, detailed place viewing, saving favorites, and exploring locations via interactive maps.

Newly implemented features further enrich the experience: language switching between English and Vietnamese, real-time weather display using OpenWeatherMap, city-specific restaurant recommendations, a quiz game to earn points and credits, and a daily login streak system stored in Firestore.

To ensure that TrekBuddy would align with real user needs and preferences, an initial exploratory user study was conducted before feature development began. This informal research involved discussions with potential users (N = 4) who frequently use travel applications. These early interviews (Appendix 2) focused on common pain points in travel planning, especially when traveling in unfamiliar cities.

Simultaneously, a market review (Appendix 3) of well-known travel apps, such as Google Travel and TripAdvisor, was conducted to identify standard and missing features. The combined insights helped shape the feature set and user experience goals of TrekBuddy.

Based on these early findings, the following user stories were created to define the initial product scope:

- As a user, I want to receive restaurant recommendations in the city I'm exploring.
- As a user, I want to know the real-time weather for each city I want to visit.
- As a user, I want to switch the app language between English and Vietnamese.
- As a user, I want to play quiz games to earn points and credit.
- As a user, I want to build a login streak and feel rewarded for daily engagement.

4.3 New Features Development

This section outlines the development of new features added to the TrekBuddy application, including multilingual support, real-time weather display, city-based restaurant recommendations, and quiz-based gamification. These additions aim to improve user engagement, personalization, and overall experience.

The implementation spans both front-end components built with React Native and back-end services powered by Firebase and third-party APIs. Real-time weather is integrated using the OpenWeatherMap API, while restaurant suggestions are dynamically retrieved via the Google Places API. Firebase Firestore manages user data such as quiz points, credit balance, and login streaks. Language switching between English and Vietnamese is supported through the i18n framework.

Each feature required UI updates for user interaction and logic for data handling or external API integration. Tools like `react-i18next`, `moment.js`, and `Firebase SDKs` were essential in enabling these functionalities.

4.3.1 Feature 1 - Multilingual Support (English – Vietnamese)

To implement this feature, I used `i18next`, which is an internationalization-framework written in and for JavaScript. (i18next, n.d.). More details and steps to work with `i18n` can be found here: https://www.i18next.com/.

First, I installed `i18n-js` and expo-localization libraries. To enable internationalization in a React Native project, we used:

```
nayoungg@MacBook-Pro ~ % npm install i18n-js expo-localization
```

Listing 11. Command line to install the required package.

Then, I created a `utils/i18n.js` file to define translation keys for both English and Vietnamese.

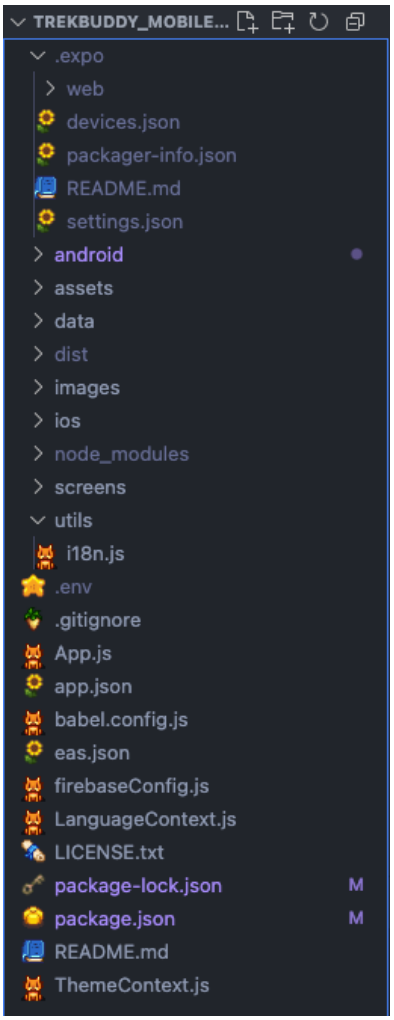


Figure 26. `i18n.js` file locates in utils folder in root file.

Figure 26 illustrates the folder and file structure of the TrekBuddy mobile application, developed using React Native. It shows how the project is organized into directories for assets, platform-specific code (Android/iOS), screens, utilities, configuration files, and key entry points such as `App.js`, `firebaseConfig.js`, and context providers for theming and language support.

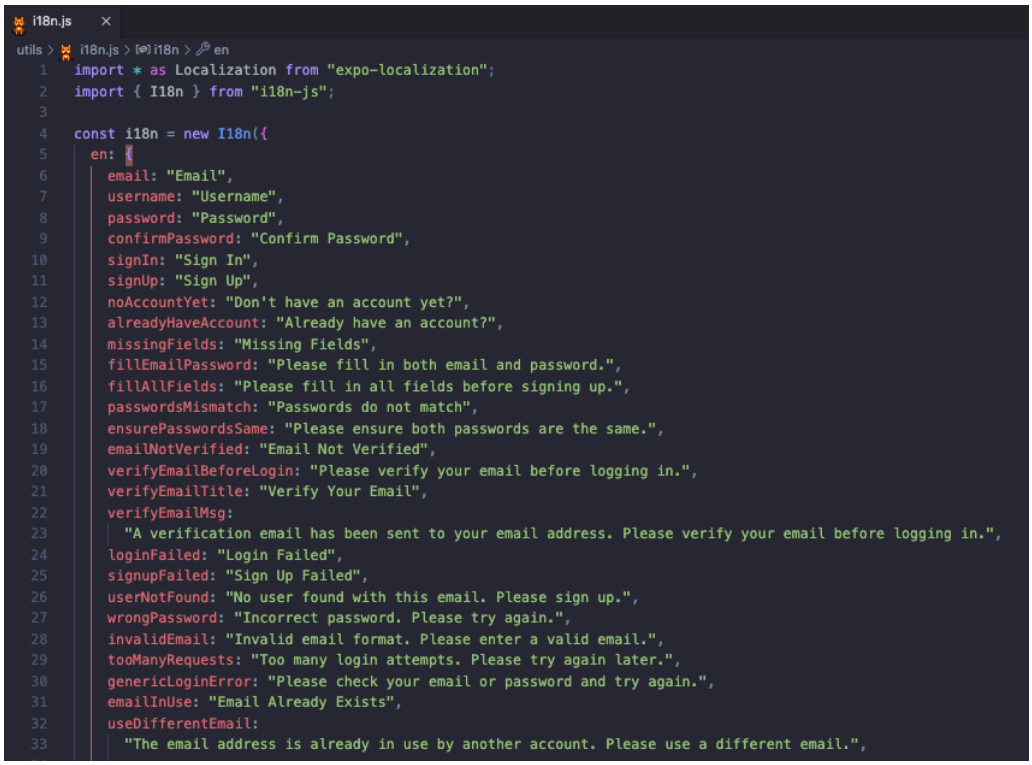


Figure 27. English localization settings in the `i18n.js` file, defining text translations for user interface elements and error messages in TrekBuddy.

Figure 27 shows the English (`en`) language translation object defined in the `i18n.js` file of the TrekBuddy mobile application. This configuration uses the i18n-js library in combination with `expo-localization` to provide dynamic, locale-based text rendering. The English translations include commonly used interface strings such as field labels (e.g., "Email", "Password"), action prompts (e.g., "Sign in", "Sign up"), and error messages (e.g., "Login failed", "Invalid email format"). These entries serve as the default fallback language for the app.

```
vi: {
  username: "Tên người dùng",
  password: "Mật khẩu",
  confirmPassword: "Xác nhận mật khẩu",
  signIn: "Đăng nhập",
  signUp: "Đăng ký",
  noAccountYet: "Chưa có tài khoản?",
  alreadyHaveAccount: "Đã có tài khoản?",
  missingFields: "Thiếu thông tin",
  fillEmailPassword: "Vui lòng nhập đầy đủ email và mật khẩu.",
  fillAllFields: "Vui lòng điền đầy đủ thông tin trước khi đăng ký.",
  passwordsMismatch: "Mật khẩu không khớp",
  ensurePasswordsSame: "Vui lòng đảm bảo cả hai mật khẩu giống nhau.",
  emailNotVerified: "Email chưa được xác minh",
  verifyEmailBeforeLogin: "Vui lòng xác minh email trước khi đăng nhập.",
  verifyEmailTitle: "Xác minh Email",
  verifyEmailMsg:
    "Email xác minh đã được gửi đến địa chỉ email của bạn. Vui lòng xác minh trước khi đăng nhập.",
  loginFailed: "Đăng nhập thất bại",
  signupFailed: "Đăng ký thất bại",
  userNotFound: "Không tìm thấy người dùng với email này. Vui lòng đăng ký.",
  wrongPassword: "Mật khẩu không đúng. Vui lòng thử lại.",
  invalidEmail: "Định dạng email không hợp lệ. Vui lòng nhập email hợp lệ.",
  tooManyRequests: "Đăng nhập quá nhiều lần. Vui lòng thử lại sau.",
  genericLoginError: "Vui lòng kiểm tra lại email hoặc mật khẩu.",
  emailInUse: "Email đã được sử dụng",
  useDifferentEmail:
    "Địa chỉ email này đã được sử dụng bởi tài khoản khác. Vui lòng sử dụng email khác.",
```

Figure 28. Vietnamese localization settings in the `i18n.js` file, defining text translations for user interface elements and error messages in TrekBuddy.

Figure 28 presents the Vietnamese (`vi`) translation object used within the same `i18n.js` file. Each English string is mapped to its Vietnamese equivalent, allowing users whose devices are set to Vietnamese to interact with the TrekBuddy app in their native language. The translated strings cover the same interface elements and validation messages as in the English set, ensuring a consistent user experience across both languages.

These above figures demonstrates how the application supports internationalization (`i18n`) by defining and managing user interface text for localization. It allows the app to easily switch languages or add additional ones in the future by providing similar objects for other locales (e.g., `vi` for Vietnamese).

On this component, import:

- `expo-localization` to detect the device's locale settings.
- `i18n-js`, a JavaScript internationalization library used to manage translations.

Then, new instances of `I18n` is created with English (`en`) and Vietnamese `(vi)` as the language configuration. In there, the `en` and `vi` objects contain **key-value pairs** where the keys (e.g., `email, signIn, missingFields`) represent identifiers used in the code and the values (e.g., `"Email", "Sign In", "Missing Fields"`) are the actual English strings shown to users.

After that, integrated the language context using React Context API. This component handling automatic detection, storage, and toggling of the app's language between English and Vietnamese.

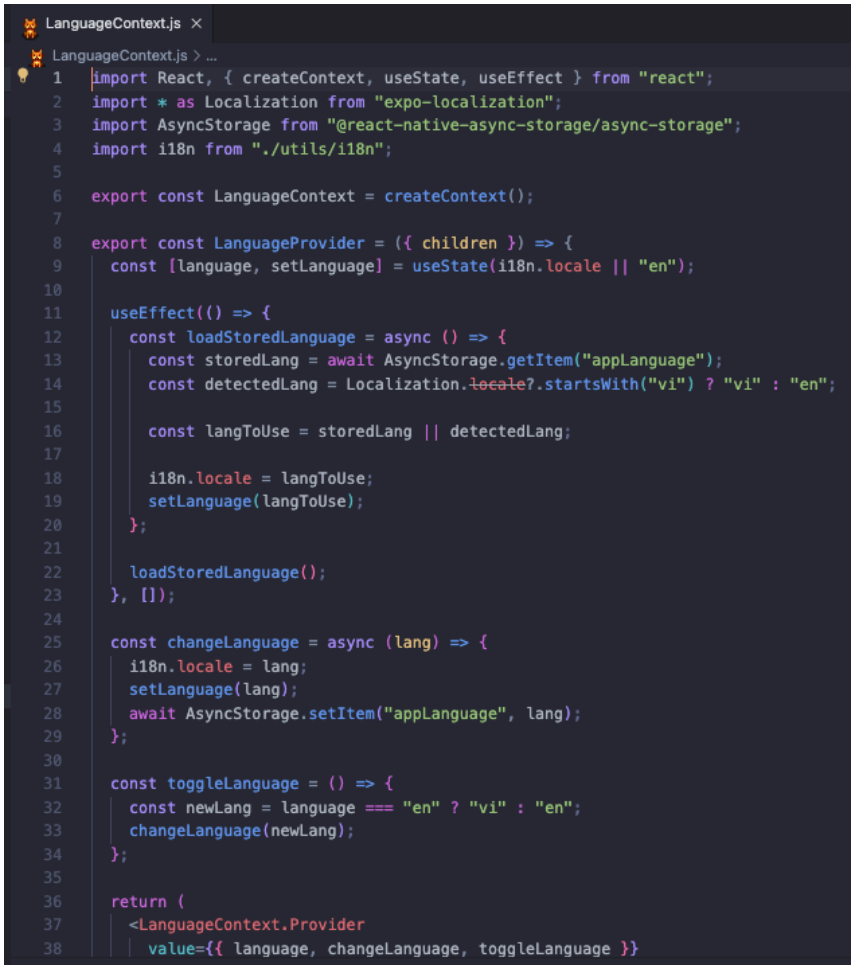


Figure 29. Language context provider implementation in **`LanguageContext.js`**

The figure illustrates the `LanguageContext.js` file, which handles language selection and switching in the TrekBuddy mobile application. It uses React's Context API to provide a global language state, enabling components across the app to access and update the current language. On initialization, it attempts to load a previously saved language from `AsyncStorage`; if none is found, it detects the device's locale using expo-localization. Based on the result, it sets the default language to either Vietnamese `("vi")` or English `("en")`, ensuring the app starts in a relevant language for the user.

The file also defines functions to change and toggle the app's language. The changeLanguage function updates the language in both the `i18n` instance and the app state, while saving the preference to storage. The `toggleLanguage` function switches between English and Vietnamese automatically. These functions, along with the current language, are provided to the rest of the app

through `LanguageContext.Provider`. This setup allows TrekBuddy to offer consistent, dynamic multilingual support throughout the user interface.

Finally, I implemented a language toggle button in the UI using `TouchableOpacity` and the `toggleLanguage` function from the language context, allowing users to switch languages dynamically. To enable localization, all static text in the interface was replaced with corresponding translation keys using `i18n.t('key')`.

```
<TouchableOpacity onPress={toggleLanguage} style={styles.iconButton}>
  <View style={styles.languageIcon}>
    <Text style={styles.languageText}>
      {language === "vi" ? "VI" : "EN"}
    </Text>
  </View>
</TouchableOpacity>
```

Figure 30. Button component for toggling app language between English (EN) and Vietnamese (VI).

This figure shows a React Native UI component for toggling the application language between Vietnamese and English. The component is built using a `TouchableOpacity` wrapper, which allows the user to press the button to trigger the `toggleLanguage` function—previously defined in the `LanguageContext`.

Inside the button, a View component contains a Text element that conditionally displays either `"VI"` or `"EN"` based on the current value of the language variable. If the language is `"vi"`, it displays `"VI"`; otherwise, it defaults to `"EN"`. This simple and intuitive UI element serves as a language switch indicator and allows users to change the app's language dynamically. Styling is applied via `styles.languageIcon`, `styles.languageText`, and `styles.iconButton`, which are assumed to handle layout and visual appearance.

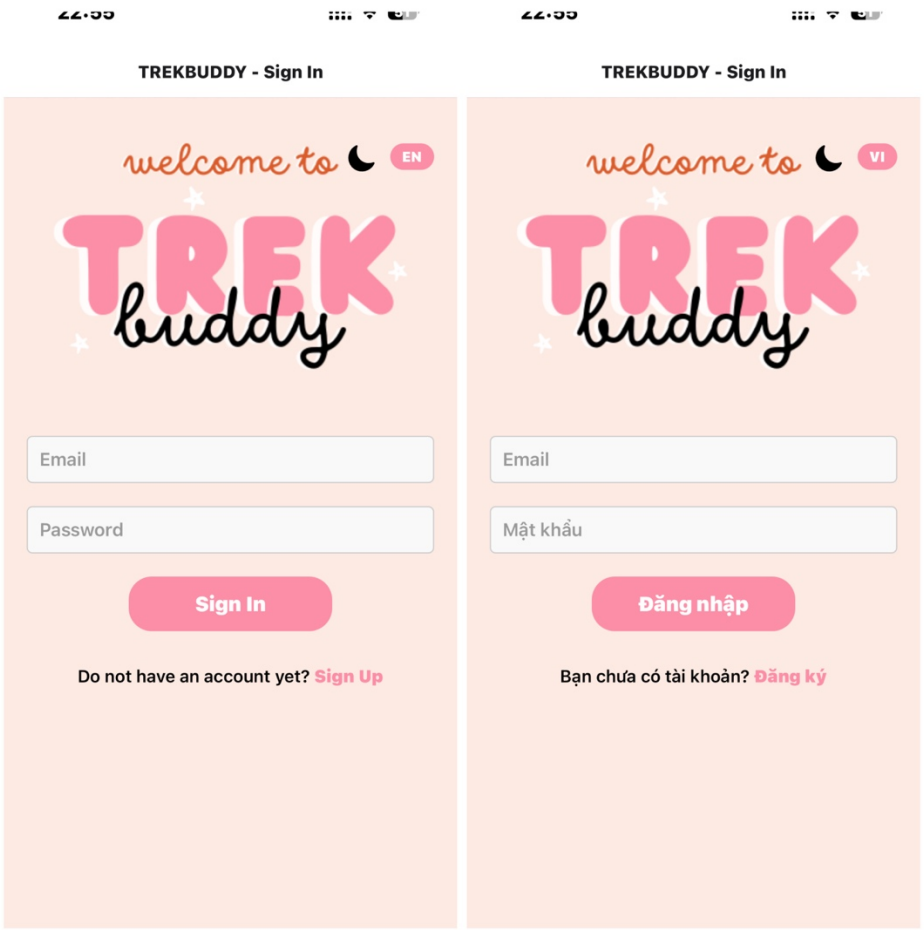


Figure 31. TrekBuddy Login page with 2 languages after implementing

Challenges, Solutions, and Learning Outcomes

During the implementation of internationalization in the TrekBuddy app using the `i18next` library, a recurring challenge was the issue of missing translation keys, such as the `name` value not appearing in the user interface. This often occurred when keys were undefined or improperly referenced in the translation configuration, leading to undefined or blank text.

To resolve this, I carefully reviewed the `i18n.js` file to ensure all necessary keys were present and correctly structured in both English and Vietnamese. I also enabled debug mode in `i18next` to identify missing keys in real time and configured a fallback language (English) to prevent blank outputs when a translation was missing. Additionally, I adopted consistent and contextual key naming (e.g., `"login.name"`), which helped improve clarity and maintainability.

Thorough testing in both languages was conducted to ensure all screens displayed correctly translated content. Through this process, I learned the importance of organized translation structures, fallback mechanisms, and rigorous testing in delivering a seamless multilingual experience.

Compared to other features I had implemented earlier—such as real-time data integration or Firebase authentication—this i18n feature was relatively straightforward. I only needed to create a single translation component, import it into the necessary screens, and replaced plain text with i18n key, making the integration process efficient and reusable across the application.

4.3.2 Feature 2 - Real-time Weather Display for each city.

First of all, the API setup is the most important step since this feature uses OpenWeatherMap API to fetch weather data based on the selected city's latitude and longitude. The API key is stored in the `.env` file same as Google API key implemented before.

Created a helper function is created to fetch and parse the weather data. The `fetchWeather` function sends a request to the OpenWeatherMap API using latitude and longitude to retrieve real-time weather data (temperature, condition, and icon) for a given city, and updates the `weatherDataByCity` state:

```
const fetchWeather = async (lat, lon, cityId) => {
  try {
    const response = await fetch(
      `https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lon}&appid=${OPENWEATHER_API_KEY}&units=metric`
    );
    const data = await response.json();

    if (data?.main && data?.weather?.[0]) {
      setWeatherDataByCity((prev) => ({
        ...prev,
        [cityId]: {
          temp: data.main.temp,
          condition: data.weather[0].main,
          icon: data.weather[0].icon,
        },
      }));
    }
  } catch (error) {
    console.error("Error fetching weather:", error);
  }
};
```

Figure 32. Screenshot of code snippet from `fetchWeather()` function.

The figure shows the `fetchWeather` function, an asynchronous JavaScript function used in the TrekBuddy mobile app to retrieve real-time weather data from the OpenWeatherMap API. It takes three parameters: latitude, longitude, and a `cityId` used to associate the weather data with a specific city in the app's state. The function sends a `fetch` request to the API, passing the

coordinates and an API key, and requests the temperature in metric units. Once the response is received, it is parsed into JSON format for further use.

If valid weather data is returned, the function updates the app's state using `setWeatherDataByCity`. It stores the current temperature, weather condition (e.g., Clear, Rain), and the corresponding weather icon for the selected city. Optional chaining is used to safely access nested data and avoid runtime errors. The function also includes basic error handling, logging any issues encountered during the fetch process. This implementation enables TrekBuddy to provide dynamic and location-specific weather updates, enhancing the user experience by helping travelers plan more effectively based on current conditions.

The function below is triggered when the user taps the **"Show Weather"** button for each city. When pressed, it triggers the `fetchWeather` function for the selected city's coordinates without triggering the parent press event.

```
<TouchableOpacity
  style={styles.showWeatherButton}
  onPress={(e) => {
    e.stopPropagation(); // prevent triggering parent TouchableOpacity
    if (item.location) {
      fetchWeather(item.location.lat, item.location.lng, item.id);
    }
  }}
>
  <Text style={styles.showWeatherButtonText}>
    {i18n.t("weather")}
  </Text>
</TouchableOpacity>
```

Figure 33. Implementation of the weather button using `TouchableOpacity`.

This figure displays a React Native component that provides a button for displaying real-time weather information for a specific location in the TrekBuddy app. The component is wrapped in a `TouchableOpacity`, which acts as a clickable element styled using `styles.showWeatherButton`. When the button is pressed, an `onPress` event handler is triggered. Inside this handler, `e.stopPropagation()` is called to prevent the event from bubbling up to any parent `TouchableOpacity`, which ensures only this button's action is executed.

If the selected item contains a valid `location` object, the `fetchWeather` function is called with the item's latitude, longitude, and ID. This fetches the weather data specific to that location. The button displays text from the app's internationalization system `(i18n.t("weather"))`, allowing it to be translated based on the selected language. This component enables users to manually

retrieve weather data for a given destination, providing control over when and where to view live weather updates in the app interface.

Once fetched, the weather data is rendered in the UI with icon and temperature. The conditional rendering of weather data in the user interface. If weather data exists, the app displays a row containing an image (sourced from OpenWeatherMap using the icon code) and a text label showing the current temperature and weather condition.:

Figure 34. JSX conditional rendering to display weather icon, temperature, and condition

This figure illustrates a React Native UI component that conditionally displays real-time weather data retrieved from the OpenWeatherMap API. The component is rendered only when the `weather` object is defined, using the `weather && (...)` syntax for conditional rendering.

The view is structured in a horizontal layout using `flexDirection: "row"` and centers its items vertically with `alignItems: "center"`. It includes an `<Image>` component that dynamically loads a weather icon from the OpenWeatherMap image service. The icon URL is constructed using a template literal with `weather.icon`, fetching a 2x resolution icon (e.g., "10d@2x.png") appropriate for the current weather condition. Next to the icon, a `<Text>` component displays the current temperature in Celsius `({weather.temp}°C)` followed by the weather condition (e.g., Clear, Rain). This compact, visually informative layout provides users with a quick, localized weather summary directly in the app interface.

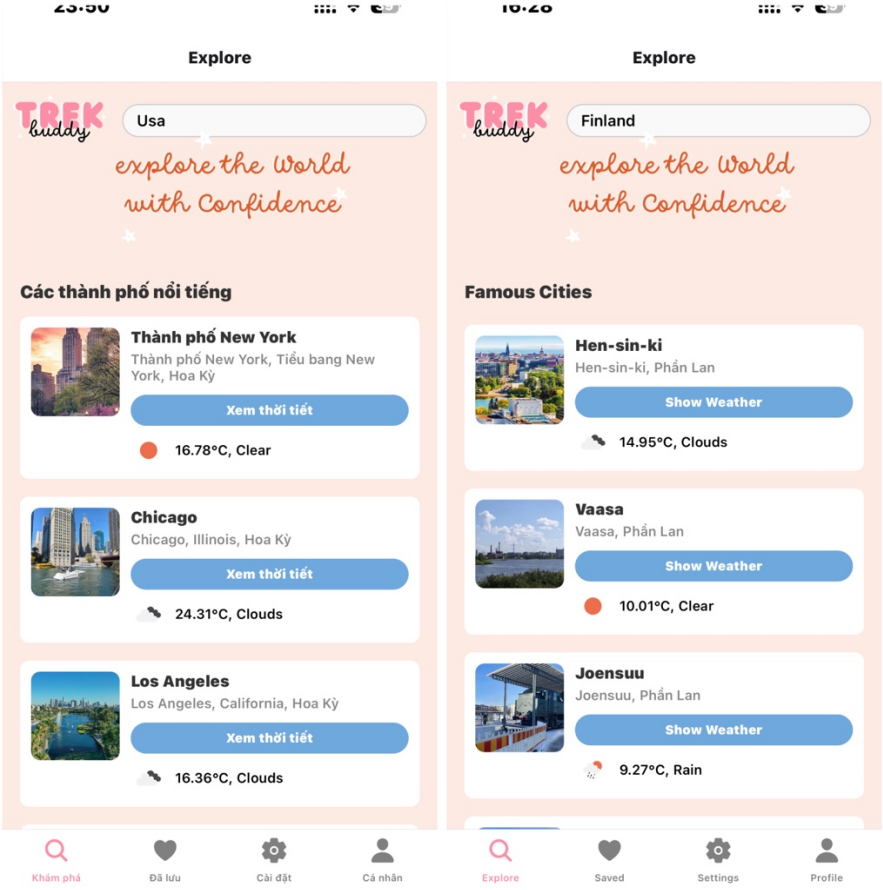


Figure 35. Real-time weather display for each city in the TrekBuddy app

Challenges, Solutions, and Learning Outcomes

The main challenge during development was rendering the correct real-time weather information for the selected city. Initially, there were issues where weather data either did not appear or was displayed for the wrong city. This was mainly due to improper state management and asynchronous function handling, which caused mismatches between the city selection and the weather data displayed.

To address this, I refined the fetchWeather function to ensure it accurately fetched data based on the selected city's coordinates. I also improved the component's state logic to correctly associate weather data with the corresponding city and updated the UI rendering conditions to prevent stale or incorrect data from being shown. Additionally, I implemented unique keys and ensured the state was reset properly between selections.

Compared to the language toggle feature, which involved static content updates and was easier to manage, the weather display feature was more complex due to its reliance on real-time data, external API integration, and asynchronous state updates.

From this process, I learned the importance of managing component-specific state and handling API responses carefully in real-time applications. I also gained practical experience in dynamic UI rendering and data synchronization between user actions and external services.

4.3.3 Feature 3 - Recommended Restaurant Listings Per City

First of all, in `HomeScreen.js`, a function is defined to fetch restaurants based on the selected city. This function that queries the Google Places API to retrieve a list of restaurants based on the selected city name, processes the results, and updates the UI with restaurant data.

```
const fetchRestaurants = async (cityName) => {
  try {
    const response = await fetch(
      `https://maps.googleapis.com/maps/api/place/textsearch/json?query=restaurants+in+${cityName}&key=${GOOGLE_API_KEY}`
    );
    const data = await response.json();
    if (data.results) {
      const restaurants = data.results.map((place) => ({
        id: place.place_id,
        name: place.name,
        address: place.formatted_address,
        location: place.geometry?.location,
        photo: place.photos
          ? `https://maps.googleapis.com/maps/api/place/photo?maxwidth=400&photoreference=${place.photos[0].photo_reference}&key=${GOOGLE_API_KEY}`
          : "https://via.placeholder.com/150",
      }));
      setRestaurants(restaurants);
    }
  } catch (error) {
    console.error("Error fetching restaurants:", error);
  }
};
```

Figure 36. Code snippet of the fetchRestaurants function

This figure shows the implementation of the `fetchRestaurants` function, an asynchronous function that retrieves restaurant data for a given city using the Google Places API. The function accepts a `cityName` as its parameter and constructs a request URL to fetch restaurant listings for that location. It uses the `textsearch` endpoint from the Places API and includes the API key stored in `GOOGLE_API_KEY`.

Once the response is fetched and parsed to JSON, the function checks if `data.results` exists. It then maps over the results to extract relevant information for each restaurant, including the `place_id`, `name`, `formatted_address`, `location`, and a photo URL. If the restaurant has a photo, it constructs the image URL using the photo reference and API key; otherwise, it defaults to a placeholder image (`https://via.placeholder.com/150`). The processed restaurant data is stored in a new array and set to state using `setRestaurants`. If any error occurs during the process, it is caught and logged using `console.error`. This function enables TrekBuddy to dynamically display restaurant recommendations based on user search queries.

Below the city title in the render section, add 2 tabs UI for switching between Destinations and Restaurants using `TouchableOpacity`. The selected tab updates the view and triggers restaurant data fetching if applicable.

```
<View style={styles.tabRow}>
  <TouchableOpacity
    style={[
      styles.tabButton,
      selectedTab === "destinations" && styles.activeTab,
    ]}
    onPress={() => setSelectedTab("destinations")}
  >
    <Text
      style={[
        styles.tabText,
        selectedTab === "destinations" && styles.activeTabText,
      ]}
    >
      {i18n.t("destinations")}
    </Text>
  </TouchableOpacity>
  <TouchableOpacity
    style={[
      styles.tabButton,
      selectedTab === "restaurants" && styles.activeTab,
    ]}
    onPress={() => {
      setSelectedTab("restaurants");
      fetchRestaurants(selectedCity);
    }}
  >
    <Text
      style={[
        styles.tabText,
        selectedTab === "restaurants" && styles.activeTabText,
      ]}
    >
      {i18n.t("restaurants")}
    </Text>
```

Figure 37. Two-tab UI component for toggling between “Destinations” and “Restaurants”

This figure displays a tab-switching user interface in React Native, used to toggle between two main views: "**destinations**" and "**restaurants**". The structure is built using `TouchableOpacity` components that act as tab buttons, styled based on the currently selected tab. The UI is wrapped inside a parent `View` with styling applied via `styles.tabRow`.

Each tab button uses conditional styling: if the `selectedTab` matches the current tab (e.g., "`destinations`" or "`restaurants`"), it applies styles.activeTab for the button and styles.activeTabText for the label to visually highlight the selected tab. Pressing the "destinations" tab sets the selected tab state without additional actions, while pressing the "restaurants" tab not

only changes the selected tab but also triggers the `fetchRestaurants(selectedCity)` function. This fetches restaurant data for the currently selected city, making the tab interactive and dynamic. Both tabs use `i18n.t(...)` for translation support, ensuring the labels are localized based on the user's selected language.

Next, render restaurant list conditionally in the section where destinations are shown, add `Flat-List` dynamically displays either destinations or restaurants.

```
<Text style={styles.sectionTitle}>
  {selectedTab === "destinations"
    ? `${i18n.t("famousDestinations")} `
    : `${i18n.t("recommendedRestaurants")} `}
</Text>

<FlatList
  data={
    selectedTab === "destinations" ? touristDestinations : restaurants
  }
  renderItem={renderDestinationItem}
  keyExtractor={(item) => item.id}
  contentContainerStyle={styles.placesList}
/>
```

Figure 38. Code snippet of conditional rendering of sections titles and content list

This figure shows a dynamic UI rendering logic in React Native that updates content based on the currently selected tab, either "**destinations**" or "**restaurants**". At the top, a `<Text>` component displays a section title using internationalized strings. It conditionally renders "famousDestinations" or "`recommendedRestaurants`" using `i18n.t()` depending on the value of `selectedTab`. This supports multilingual display of the section title.

Below the title, a `<FlatList>` component is used to display a scrollable list of items. The list's data source dynamically switches between `touristDestinations` and `restaurants` arrays based on the same `selectedTab` value. The `renderItem` prop determines how each item in the list is displayed, while `keyExtractor` ensures each item has a unique key based on its `id`. This pattern allows for a clean, reusable interface that adapts to user interaction and supports localization, contributing to a responsive and context-aware user experience in the TrekBuddy app.

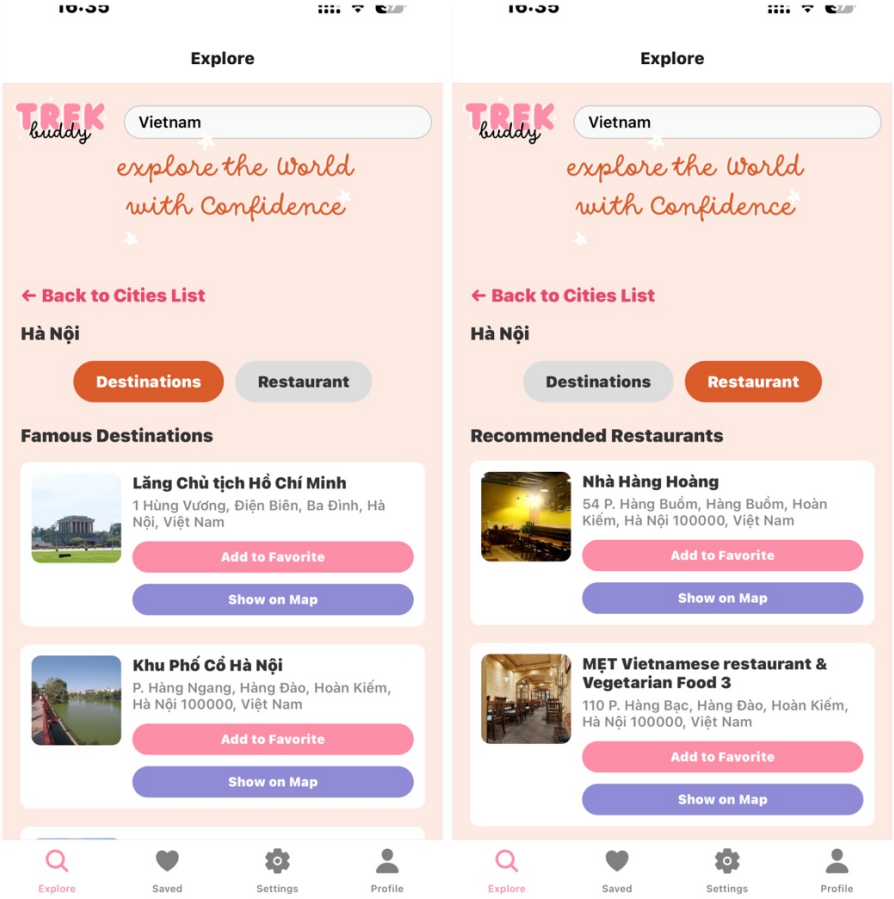


Figure 39. User interface showcasing recommended destinations and restaurants in Hanoi, Vietnam.

Challenges, Solutions, and Learning Outcomes

To implement the feature, a `fetchRestaurants` function was created in `HomeScreen.js`. This function queries the Google Places API based on the selected city and stores the resulting data in state. Below the city title, a two-tab interface using `TouchableOpacity` allows users to switch between Destinations and Restaurants. Depending on the selected tab, a `FlatList` dynamically renders the relevant content.

The main issue occurred when trying to update and render the correct list based on user selection. Initially, the UI either failed to refresh correctly or displayed mismatched content. This was resolved by improving the tab state logic, clearly separating destination and restaurant data states, and applying conditional rendering based on the selected tab. I also ensured the `FlatList` keys and data sources were reset properly when toggling views to avoid rendering glitches.

Despite the rendering challenges, having previously implemented a similar destination fetching feature made this development more manageable. Reusing logic patterns from the destination component helped streamline the data-fetching process.

From this experience, I learned how to better manage multiple dynamic lists within the same screen and the importance of clear state separation when dealing with conditional UI components. This feature reinforced my understanding of state-driven rendering and component reusability in React Native applications.

4.3.4 Feature 4 - Quiz Game to Earn Points and Credits.

First, I created `QuizScreen.js` to handle question display, answer validation, score tracking, and Firestore updates.

Figure 40. Code snippet from `QuizScreen.js`

The figure displays the implementation of the `QuizScreen` component in the TrekBuddy mobile application, which provides users with a gamified quiz experience. It utilizes React's state management to track the current question, user score, selected answer, and whether the user has already answered. When a user selects an answer, the `handleAnswer` function checks if the option is correct by comparing it to the predefined answer for the current question. If correct, the score is incremented and the `awardCredit` function is triggered to update the user's points in Firebase Firestore. Feedback is provided immediately using localized alert messages through the `i18n` internationalization system.

The `awardCredit` function accesses the user's Firestore document to update their total quiz points. It first retrieves the document using the current user's ID, checks if it exists, and either updates the existing points or initializes them if not present. The component also includes a `nextQuestion` function, which advances the quiz if there are remaining questions or shows a final alert with the user's score when the quiz is completed. The navigation is then reset to return the user to the start of the quiz. This structured approach provides a seamless, engaging, and educational experience that encourages users to return daily and interact with the app's content.

By integrating quiz functionality with real-time score tracking and Firebase-based persistence, the `QuizScreen` contributes significantly to TrekBuddy's gamified experience. The use of multilingual support through `i18n`, responsive state handling, and interactive UI logic reflects thoughtful design that aligns with the app's broader goals of engagement and accessibility.

Then, added Navigation to Quiz from ProfileScreen since users can see their progess including Credits and Points from ProfileScreen, so the CTA button to ask users to play quiz and earn points is located in the ProfileScreen.

```
<TouchableOpacity
  style={styles.quizButton}
  onPress={() => navigation.navigate("QuizScreen")}
>
  <Text style={styles.quizButtonText}>{i18n.t("takeQuiz")}</Text>
</TouchableOpacity>
```

Figure 41. Navigation button on the Profile screen that allows users to start the quiz game by navigating to `QuizScreen`.

This figure shows a React Native component that provides a button for navigating to the quiz feature within the TrekBuddy app. The `TouchableOpacity` element is styled with `styles.quizButton` and triggers navigation to the QuizScreen when pressed. The button's label is rendered using a `Text` component and supports internationalization through

`i18n.t("takeQuiz")`, allowing the displayed text to adapt based on the selected language. This component serves as an entry point for users to access the quiz functionality, contributing to the app's gamification and user engagement features.

Updated Firestore user document with 2 fields: **credits** and **points** to reflect user performance. Additionally, fetch the current credits and points from Firestore to display on Profile page for each user.

```
useEffect(() => {
  const fetchUserData = async () => {
    try {
      const userDocRef = doc(db, "Users", auth.currentUser.uid);

      const userDoc = await getDoc(userDocRef);
      if (userDoc.exists()) {
        const userData = userDoc.data();
        setUsername(userData.username || "Unknown");
        setPhotoURL(userData.photoURL || null);
        setCredit(userData.credit || 0);
        setPoints(userData.points || 0);
        setStreak(userData.streak || 0);
      } else {
        await setDoc(userDocRef, {
          username: "Unknown",
          photoURL: null,
        });
      }
    } catch (error) {
      setErrorMessage(i18n.t("fetchUserFail") + error.message);
    }
  };
  fetchUserData();
}, []);
```

Figure 42. Function `fetchUserData()` used to retrieve user information including points and credits from Firestore.

This figure shows a `useEffect` hook in React that retrieves user data from Firebase Firestore when the component is mounted. The `fetchUserData` asynchronous function accesses the currently authenticated user's document from the `Users` collection using their UID. If the document exists, it extracts and sets various user-related states such as `username`, `photoURL`, `credit`, and `points`. Default values are provided in case any field is missing—for example, `"Unknown"` for usernames or `0` for numerical values like points or streaks. This ensures that the UI displays consistent and valid data even when some fields are uninitialized.

If the user document does not exist, the function creates a new document with default values for `username` and `photoURL` to initialize the user profile in Firestore. Any errors encountered during

the fetch process are caught and stored in an `errorMessage` state, with the message also translated using `i18n.t("fetchUserFail")` for localization. The `fetchUserData` function is called once upon component mount due to the empty dependency array in the `useEffect`. This logic ensures reliable and localized user data retrieval and handling, forming the basis for personalized features and display throughout the TrekBuddy application.

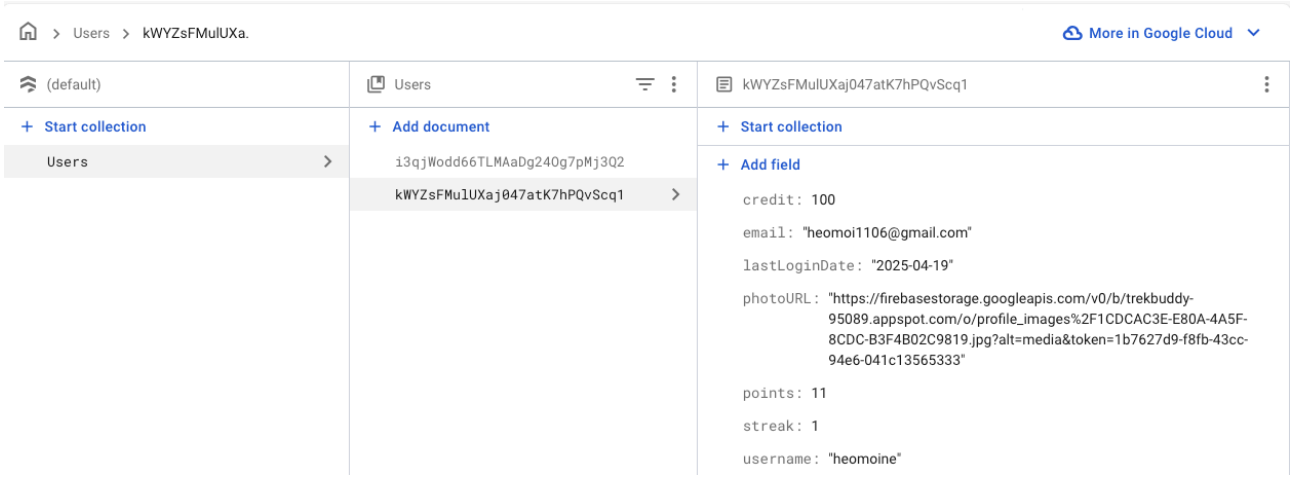


Figure 43. Firestore database showing user document structure including points, credit fields.

This figure shows a Firebase Firestore document within the `Users` collection, representing a specific user identified by the document ID `kWYZsFMulUXaj047aTk7hPQvScq1`. The document stores key user data including `username`, `email`, `lastLoginDate`, `photoURL`, `credit`, and `points`. These fields are used in the TrekBuddy app to personalize the user experience, track quiz performance, and display profile information.

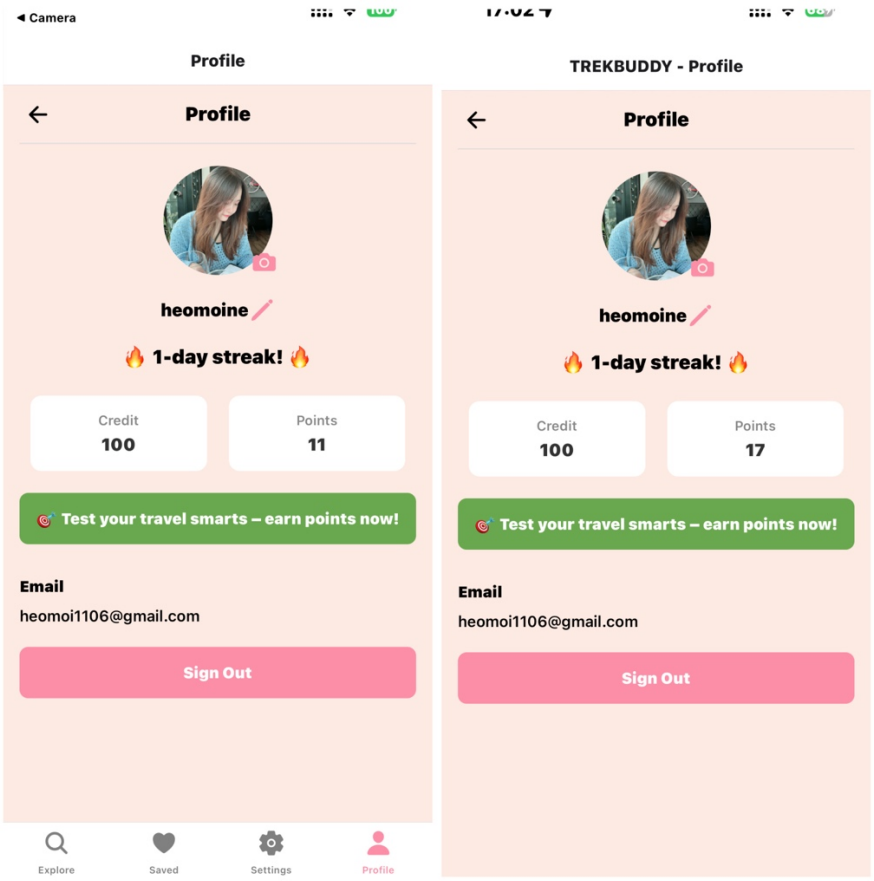


Figure 44. Profile screen before and after the quiz, showing updated points after completing the game.

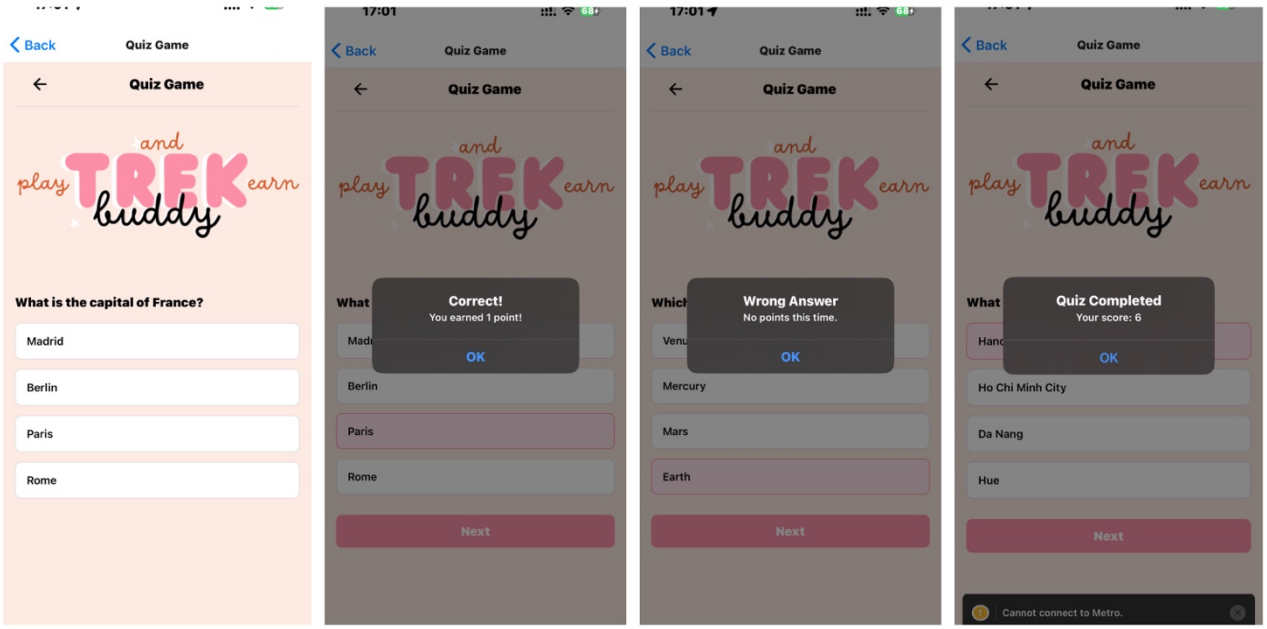


Figure 45. Quiz game flow: answering a question, getting feedback for correct or incorrect answers, and viewing final score after completing the quiz.

Challenges, Solutions, and Learning Outcomes

The main challenge encountered during implementation was that the user's score did not update correctly in Firestore after completing the quiz. Additionally, even when data appeared to be written, the app failed to fetch and display the updated `credits` and `points` from Firestore on the ProfileScreen. This issue disrupted the intended feedback loop and undermined the gamified experience.

To address this, I first debugged the Firestore write operation within the `QuizScreen.js` file. I ensured that the `updateDoc` function targeted the correct user document and that asynchronous operations were handled properly using `await`. Secondly, I reviewed the logic in the ProfileScreen to confirm that the data fetching from Firestore used the correct user ID and that the fetch was triggered after quiz completion. I also verified that state updates occurred after successful reads.

Once resolved, the feature functioned as intended: after finishing the quiz, the user's score was recorded in Firestore and immediately reflected in the Profile screen upon navigation. A navigation button was placed on the ProfileScreen to encourage users to play and earn more points, reinforcing user engagement.

Compared to other features such as the **real-time weather display**, which involves one-way data fetching, this quiz feature introduced a two-way data flow: writing to and reading from the database. This made it more complex and required a careful approach to asynchronous operations and state management.

From this feature, I learned the importance of verifying data flow in both directions when working with cloud databases like Firestore. I also gained experience in handling async/await logic, managing state updates in response to remote changes, and creating interactive components that tie closely to personalized user data.

4.3.5 Feature 5 – Daily Login Streak

To increase user retention and encourage daily app usage, a daily login streak feature was implemented in the LoginScreen. This system tracks the number of consecutive days a user logs into the app and updates their streak accordingly in Firebase Firestore. The feature uses the `moment.js` library to compare the current login date to the last recorded login date and determine if the streak should continue or reset.

```javascript
const updateLoginStreak = async () => {
  const userRef = doc(db, "Users", auth.currentUser.uid);
  const userSnap = await getDoc(userRef);

  if (userSnap.exists()) {
    const data = userSnap.data();
    const today = moment().format("YYYY-MM-DD");
    const yesterday = moment().subtract(1, "day").format("YYYY-MM-DD");
    const lastLoginDate = data.lastLoginDate || null;
    let newStreak = 1;

    if (lastLoginDate === today) {
      console.log("Already logged in today, no change to streak.");
      return;
    } else if (lastLoginDate === yesterday) {
      newStreak = (data.streak || 0) + 1;
    }

    await updateDoc(userRef, {
      streak: newStreak,
      lastLoginDate: today,
    });
    console.log(`Streak updated: ${newStreak}`);
  } else {
    console.log("User document does not exist.");
  }
};
```

Figure 46. Code snippet for updating the login streak in Firebase Firestore.

This figure shows the `updateLoginStreak()` function, which is called after successful user authentication. The function retrieves the user document from the Firestore Users collection and checks the `lastLoginDate` field. If the user has already logged in on the current day, no changes are made. If the last login was exactly one day ago, the `streak` count is incremented by one. If there is no valid login streak or a day was missed, the streak resets to 1. Finally, the new streak and today's date are written back to Firestore to persist the updated state.

```
const handleLogin = async () => {
    if (!email || !password) {
      Alert.alert(i18n.t("missingFields"), i18n.t("fillEmailPassword"));
      return;
    }

    try {
      const userCredential = await signInWithEmailAndPassword(
        auth,
        email,
        password
      );
      const user = userCredential.user;

      if (!user.emailVerified) {
        Alert.alert(
          i18n.t("emailNotVerified"),
          i18n.t("verifyEmailBeforeLogin")
        );
        await auth.signOut();
        return;
      }

      console.log("User logged in:", user);
      await updateLoginStreak();
      navigation.navigate("HomeScreen");
      }
```

Figure 47. Login logic calling `updateLoginStreak` function after successful sign-in.

This figure presents the login logic that triggers the login streak update. Once a user is authenticated and their email is verified, the app executes `updateLoginStreak()` before navigating to the HomeScreen. This ensures that the user's streak is checked and updated in real time during every valid login session. Additionally, localized alerts are used to display messages in the appropriate language (e.g., missing fields, unverified email) through the i18n internationalization framework.

The user document is updated with two fields:

- `streak`: the number of consecutive days the user has logged in
- `lastLoginDate`: the most recent login date in "YYYY-MM-DD" format

These values are used both to maintain the streak logic and to personalize the user profile, displaying the current streak as a 🔥 icon and count on the ProfileScreen.

```
credit: 100
email: "heomoi1106@gmail.com"
lastLoginDate: "2025-05-01"
photoURL: "https://firebasestorage.googleapis.com/v0/b/trekbuddy-
          95089.appspot.com/o/profile_images%2Ff9869d4a-74af-45a6-a663-
          59c586e07073.jpeg?alt=media&token=9463fe61-efb0-46a9-80c5-
          44208d6e5907"
points: 55
streak: 2
username: "Heomoinee"
```

Figure 48. Firestore user document with streak and lastLoginDate fields

This figure shows a sample Firestore document for a user. The document includes key fields like username, email, credits, and now the login streak. These fields are updated each time the user logs in, allowing the app to show accurate progress indicators.

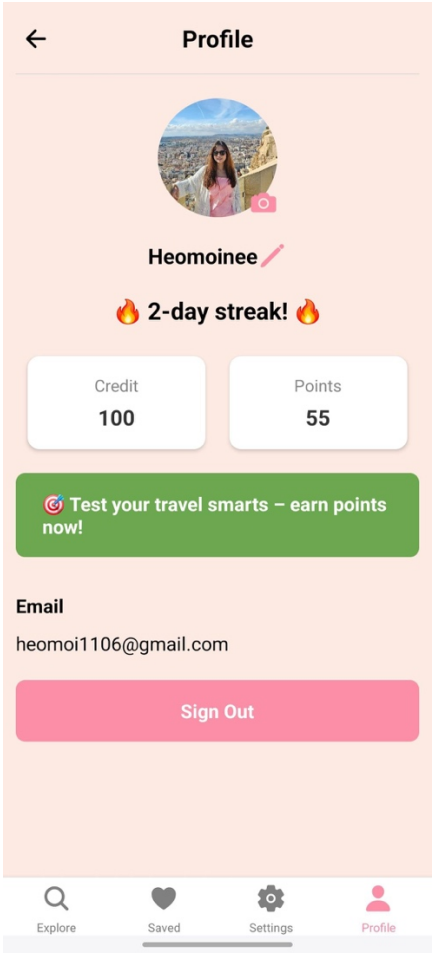


Figure 49. Profile screen displaying user's current login streak with streak icon

4.4 New Features Testing

4.4.1 Testing Environment and Methodology

Testing was conducted using the Expo Go environment on an iPhone 14 Pro simulator to simulate real-world mobile usage conditions. Even though testing on Expo Go simulator has many limitations, the testing code using the testing framework (e.g., Jest) can be added in the future. This is mentioned in Chapter 6, section 6.2.2. A scenario-based manual testing methodology was adopted. Each scenario reflected a complete user journey involving multiple features, including language selection, city exploration, real-time weather retrieval, restaurant discovery, and user engagement through a quiz game.

All test cases were executed manually and documented using a structured format, capturing the input actions, expected outcomes, and final results. The goal was to identify any issues that could affect user interaction across different modules of the application.

4.4.2 End-to-End Feature Testing

Multilingual (EN – VI) Functionality

The multilingual functionality was tested to confirm that the application correctly supported language switching between English (default) and Vietnamese (localized). The testing objectives included verifying real-time language toggling, translation consistency, fallback behavior, language persistence across screens, and UI adaptability to different text lengths.

Testing began by launching the application and interacting with the language toggle button. The app was expected to dynamically switch text content upon tapping the toggle without requiring a restart. Specific text elements, such as buttons and screen headers, were verified for accurate translation. For example, the English "Sign In" button was correctly translated to "Đăng nhập" in Vietnamese.

To test fallback mechanisms, certain keys were intentionally removed from the Vietnamese translation file. Upon switching to Vietnamese, the application successfully displayed the English fallback values, validating the reliability of the fallback logic provided by the i18next library.

Furthermore, screen navigation tests were conducted to confirm language persistence across the application. After changing the language on the login screen, transitions to the Home and Profile screens retained the selected language setting. Additionally, visual inspection confirmed that the UI layout remained intact despite the increased text length in Vietnamese, demonstrating proper layout flexibility and internationalization readiness.

Real-Time Weather Display

The real-time weather feature was tested to ensure that the application could successfully retrieve and present live weather data for individual cities based on their geographical coordinates. The testing scenario simulated a typical user workflow: navigating to the list of cities, selecting a city, and pressing the “Show Weather” button.

The application was expected to query the weather API and display relevant weather information such as temperature and weather condition. The results were confirmed visually within the app interface. The weather data was loaded promptly and displayed accurately, indicating a successful integration between the application frontend and the weather API backend.

Restaurant Listing

Testing of the restaurant listing feature was designed to verify that restaurant data was correctly fetched, displayed, and navigated by users. Upon switching to the “Restaurants” tab from the city screen, the application displayed a list of restaurants with corresponding images, names, and addresses. Each entry also included an interactive “Show on Map” button.

Functionality was verified by tapping the map button, which opened a focused map view centered on the selected restaurant’s geolocation. In addition, tab-switching performance was tested by alternating between the “Destinations” and “Restaurants” tabs. The content updated seamlessly, and no UI issues were observed.

Error handling was also evaluated by disabling the device's internet connection and attempting to load the restaurant list. The application logged the error gracefully and avoided crashing, which confirmed that appropriate fallback logic and resilience were in place for offline conditions.

Quiz Game

The interactive quiz game was tested to assess its ability to deliver an engaging user experience, handle user input, and update user scores in real time using Firebase Firestore. The test began by navigating from the Profile screen to the Quiz screen, where a set of questions with multiple-choice answers was presented.

The quiz logic was validated by selecting both correct and incorrect answers. A correct answer resulted in a success message and score increment in Firestore, while an incorrect selection triggered a “Wrong” message without score modification. This verified the correctness of the conditional feedback and scoring mechanisms.

Users were able to proceed through multiple questions using a “Next” button until the quiz completion screen appeared, showing the final score. The Profile screen was refreshed post-quiz to confirm that the updated score was accurately reflected in the user’s profile data, verifying that backend integration with Firebase was functioning correctly.

4.4.3 Results and Observations

The integrated testing of the TrekBuddy application confirmed that the app operates reliably across all major features and interaction flows. The system demonstrated excellent handling of user inputs, seamless screen transitions, dynamic content rendering, and consistent language support.

Test Case Description	Input/Action	Expected Outcome	Status
Verify language toggle button switches language	Tap on the language toggle icon (EN/VI)	All text content changes between English and Vietnamese immediately	Passed
Check translation for `signIn` button	Switch language to Vietnamese	"Sign In" becomes "Đăng nhập"	Passed
Ensure fallback works if a translation key is missing	Remove a key (e.g., `points`) from Vietnamese dictionary and switch to VI	The English version ("Points") is displayed as fallback	Passed
Verify language persists during navigation	Toggle language and navigate from Login → Home → Profile	Toggle language and navigate from Login → Home → Profile	Passed
Confirm UI layout handles text length differences	Switch between languages on all screens	No UI distortion or text overflow occurs	Passed
To verify that the app successfully fetches and displays real-time weather information for each city	Locate a city item and tap the **"Show Weather"** button.	Display the real-time weather data	Passed

Restaurant list displays correctly	Select a city and switch to ”Restaurants” tab	A list of restaurants with image, name, address, and buttons is shown.	Passed
Switch between tabs	Tap "Destinations" and "Restaurants" tabs repeatedly.	Correct content is shown based on the selected tab.	Passed
Restaurant "Show on Map" works	Tap "Show on Map" on a restaurant item.	Map opens and focuses on the restaurant's location.	Passed
Restaurant list error handling	Turn off internet connection and fetch restaurants.	Error is logged in console; no app crash.	Passed
Open Quiz Game	Navigate from ProfileScreen to QuizScreen using CTA button	Quiz interface displays with questions and options	Passed
Answer Correctly	Select correct answer	Score increases, point added in Firestore, "Correct" alert shown	Passed
Answer Incorrectly	Select wrong answer	No score change, "Wrong" alert shown	Passed
Navigate All Questions	Press "Next" after each question	Moves to the next question until complete	Passed
Finish Quiz	Reach end of question list	Show total score and navigate back to ProfileScreen	Passed
Refresh Profile	Return to ProfileScreen	Updated `points` displayed correctly	Passed

Table 1. Testing results and observations.

4.5 Deployment Using EAS

4.5.1 Rationale for Choosing EAS for Deployment

EAS (Expo Application Services) was selected to deploy the TrekBuddy app because it integrates smoothly with React Native and supports projects developed using Expo Go. Since TrekBuddy was built and tested using Expo Go, EAS was a natural choice, allowing for simple configuration, local or cloud builds, and native signing without complex Gradle setup.

I followed the EAS Build guideline to deploy my TrekBuddy app in local to create apk file, avoid waiting time in the queue in EAS Cloud. Deploy using EAS guideline can be found at: https://docs.expo.dev/build/setup/.

4.5.2 Preparation Steps for Deployment

Before building and deploying the application, several preparations were necessary:

- Clean up the code by the removal of unused dependencies, here is the `react-native-pdf-lib` in my project to reduce build size and potential conflicts.
- Double-check that the app.json file was configured with appropriate values for `name`, `slug`, `version`, and `orientation`.
- Installed Android platform tools (e.g., adb) via Homebrew to support testing on a physical Android device.

4.5.3 EAS Build and Configuration Process

After preparations, I installed Expo CLI and EAS CLI using the command below:

```
nayoungg@MacBook-Pro ~ % npm install -g eas-cli
```

Listing 13. Command line to install EAS CLI

Then, to configure an Android project for EAS, I run the following command. This will create an `eas.json` file, which was modified to define a `production` profile using the `apk` or `aab` format.

```
nayoungg@MacBook-Pro TrekBuddy_mobile_app_project % eas build:configure
```

Listing 14. Command line to configure Android project.

Figure 50. Screenshot of code snippet in `eas.json` file.

Finally, I run the local build command to create the `.aab` file.

```
nayoungg@MacBook-Pro TrekBuddy_mobile_app_project % eas build --platform android --local
```

Listing 15. Command line to build local `.aab` file.

During the process of deploying the TrekBuddy application locally using EAS Build, I encountered several technical challenges that significantly contributed to my understanding of Android build systems and project configuration in React Native.

Initially, upon running the local build command, the process failed with the following error:

`"Could not determine the dependencies of null > Could not resolve all dependencies for configuration 'classpath'. The new Java toolchain`

`feature cannot be used at the project level in combination with source and/or target compatibility."`

Despite researching online and consulting AI-based tools, no immediate solution was found. Eventually, after discussing the issue with my advisor, the problem was resolved by adding a custom Android build profile to the eas.json configuration file:

```
"android": {
  "image": "latest"
},
```

Figure 51. Code snippet of build configuration part.

This addition ensured the project used a compatible and up-to-date build environment.

However, a subsequent error occurred:

`"Task 'clean' not found in root project 'TrekBuddy' and its subprojects."`

This indicated that the Gradle script was missing a task definition required by the build system. After further investigation and testing, I resolved the issue by manually creating a build.gradle file in the root directory and including the necessary configuration to define the missing clean task.

```
build.gradle > ...
1   task clean(type: Delete) {
2       delete rootProject.buildDir
3   }
```

Figure 52. Code snippet in /build.gradle file.

These experiences enhanced my ability to debug and configure Android builds within the React Native ecosystem. I also learned the importance of understanding build tooling and how to adapt project configurations to meet specific deployment requirements. The process ultimately reinforced the value of iterative problem-solving and seeking support when working with complex development tools.

After resolving the Gradle script issue, I executed the local build command once more. This time, the build process completed successfully without any further errors. The `.aab` (Android App Bundle) file was generated and saved to the specified directory, marking the completion of the local deployment process.

```
Build successful
You can find the build artifacts in /Users/nayoungg/Nayoungie/Tài liệu học/2024/2nd semester/Mobile Programming/TrekBuddy_mob
_app_project/build-1745492269144.aab
```

Figure 53. Successfully local build message.

Next, I need to convert `.aab` to `.apk` for installing to Android phone for testing. I installed and used `bundletool`, which is an official Google tool for converting.

```
nayoungg@MacBook-Pro ~ % java -jar ~/Downloads/bundletool-all-1.18.1.jar build-apks \
--bundle=~/Downloads/trekbuddy.aab \
--output=~/Downloads/trekbuddy.apks \
--mode=universal \
--ks=~/trekbuddy.jks \
--ks-key-alias=anhnguyen \
--ks-pass=pass:heomoi1106 \
--key-pass=pass:heomoi1106
```

Listing 16. Command line to use `bundletool` for converting.

Then extract the .apk file from the .apks ZIP by using the following command:

```
nayoungg@MacBook-Pro ~ % unzip ~/Downloads/trekbuddy.apks -d ~/Downloads/trekbuddy_apks

Archive:  /Users/nayoungg/Downloads/trekbuddy.apks
 extracting: /Users/nayoungg/Downloads/trekbuddy_apks/toc.pb
 extracting: /Users/nayoungg/Downloads/trekbuddy_apks/universal.apk
```

Listing 17. Command line to extract .apk file.

Then changed the universal.apk file to trekbuddy-v1.apk and installed it to my Android phone using a USB cable and the following command. Now the TrekBuddy app should appear on my Android phone.

```
nayoungg@MacBook-Pro ~ % adb install ~/Downloads/trekbuddy_apks/trekbuddy-v1.apk
Performing Streamed Install
Success
```

Listing 18. Command line to install trekbuddy app to Android phone.

Following the successful local build, the TrekBuddy application is now available for testing on Android devices only through the installation of the generated `.apk` file. This enables end users to interact directly with the app, providing an opportunity to evaluate its functionality, usability, and performance on actual devices. By distributing the `.apk` file for testing, I am able to gather valuable feedback from users regarding their experience with the application. These insights will be instrumental in refining the app and will be documented and analyzed in the Chapter 5 of this thesis.

5 Application Implement Results and Evaluation

This chapter presents the results of the implementation phase and evaluates the application based on user feedback, technical performance, and personal learning outcomes. The evaluation includes a user survey, app functionality testing, and reflections on development challenges and achievements.

5.1 Implementation Summary

By the end of the development phase, the following features were successfully implemented into the TrekBuddy mobile application:

- Real-time weather display using OpenWeather API
- Restaurant recommendations using Google Places API
- Multilingual support (English and Vietnamese) via i18n
- Quiz-based gamification system with Firestore integration
- Daily login streak with Firestore integration

These features were evaluated through both technical testing and user feedback to determine their usability, effectiveness, and overall contribution to the user experience.

5.2 Success Indicators and Key Outcomes

The success of TrekBuddy was measured by evaluating user satisfaction, usability, and perceived usefulness of five newly implemented features: multilingual support, real-time weather display, recommended restaurants, quiz-based gamification, and streak tracking.

To evaluate the effectiveness and usability of the newly implemented features in the TrekBuddy mobile application, a user survey was conducted between March 30 and April 11, 2025, involving 6 participants. The survey assessed five key features: multilingual language switching, real-time weather display, recommended restaurants, quiz-based gamification, and daily login streak tracking using a 5-point Likert scale, where 1 indicated "Very Easy" and 5 indicated "Very Difficult" for application navigation testing and 1 indicated “Very Satisfied” and 5 indicated “Very Dissatisfied” for new features satisfaction (Figure A14, Figure A15). It also included one open-ended question to collect suggestions for future development.

Although the sample size was small, the results provide valuable early insights into user experience and preferences. Below is a summary of the survey structure and the associated success indicators:

Q#	Question	Purpose	Linked Success Indicator
Q1	How easy was it to navigate the new features in TrekBuddy?	Evaluate overall usability	App usability and user experience
Q2	Was switching between English and Vietnamese smooth and intuitive?	Measure i18n clarity	Multilingual accessibility
Q3	Did you experience any translation issues or unclear phrases?	Identify i18n flaws	Translation quality and localization accuracy
Q4	Was the weather information displayed quickly and accurately when viewing a city?	Measure speed and accuracy	Real-time API performance and reliability
Q5	Was the weather information useful for your travel planning?	Measure relevance	Contextual usefulness and trip planning support
Q6	Did you find the recommended restaurants relevant to your selected city?	Assess location accuracy	Location-based content accuracy
Q7	Would you like to see additional restaurant details (e.g., ratings, hours)?	Identify improvement needs	Feature depth and information completeness
Q8	Was the quiz feature engaging and enjoyable to play?	Measure engagement	Gamification and user interaction
Q9	Was it clear how points or credits are awarded after each quiz?	Assess reward system clarity	Feedback transparency and motivation

Q10	Did the daily login streak feature motivate you to open the app more often?	Measure engagement incentive	Retention and habit formation
Q11	Was the streak display (e.g., 🔥 3-day streak) clear and rewarding?	Evaluate UI clarity	Motivation through visual feedback
Q12	How satisfied are you with the new features overall?	General satisfaction	Overall feature success and UX impact
Q13	What improvements or additional features would you like to see in the future?	Gather user suggestions	Future roadmap and user-centered design input

Table 2. Survey questions to measure user satisfaction when testing TrekBuddy.

Below are the results based on survey data with detailed clarification and analysis.

5.2.1 Usability and Interface Clarity

Indicators: Navigation clarity and intuitiveness (Q1)

Linked Figures: See Appendix 1, Figure A1

Outcomes:

- Q1 (*How easy was it to navigate the new features in TrekBuddy?*) received a mix of ratings: 33.3% rated the navigation as Very Difficult, 50% as Easy, and 16.7% as Very Easy.
- As shown in Figure A1, the majority (4 out of 6 users) found the app easy to navigate.
- Two users selected Very Difficult, indicating that while the UI was generally clear, there may still be room for minor improvements.
- Feedback suggests that the navigation structure met user expectations, especially for new features such as weather and quiz access.

Feedback Sample:

“Everything felt smooth. I didn’t need a tutorial to find things.”

5.2.2 Language Switching (i18n)

Indicators: Smooth toggle experience (Q2), absence of translation errors (Q3)

Linked Figures: See Appendix 1, Figure A2 and Figure A3

Outcomes:

- Q2 (*Was switching between English and Vietnamese smooth and intuitive?*) and Q3 (*Did you experience any translation issues or unclear phrases?*) both scored 100% positive responses—users agreed that language switching was intuitive and no unclear phrases were encountered.
- As shown in Figures A2 and A3, all users were able to switch between English and Vietnamese without confusion or technical issues.
- These results confirm that internationalization (i18n) was implemented effectively using react-i18next.

Feedback Sample:

“Translations were clear and switching was fast. No issues.”

5.2.3 Real-Time Weather Display

Indicators: Display accuracy and usefulness for travel planning (Q4–Q5)

Linked Figures: See Appendix 1, Figure A4 and Figure A5

Outcomes:

- Q4 (speed and accuracy) received 100% Yes responses; Q5 (usefulness) scored 83.3% Yes, and 16.7% No.
- As seen in Figures A4 and A5, the feature performed well technically but had slightly mixed perceived value for planning, depending on individual usage.
- This suggests the integration with the OpenWeather API met performance goals, but could be enhanced with additional forecasting layers.

Feedback Sample:

“I saw it would rain later, so I changed plans to visit a museum.”

5.2.4 Recommended Restaurants

Indicators: Relevance of suggestions (Q6), demand for additional data (Q7)

Linked Figures: See Appendix 1, Figure A6 and Figure A7

Outcomes:

- Q6 and Q7 both received 100% Yes responses—users confirmed that restaurant listings matched their selected cities and wanted to see more information such as hours, ratings, or reviews.
- As shown in Figures A6 and A7, this feature was positively received and considered useful, but users requested content depth beyond basic listings.
- This feedback supports the development of the user review feature allowing users to leave comments and rate locations.

Feedback Sample:

“Would love to see ratings and real user reviews before choosing.”

5.2.5 Quiz-Based Gamification

Indicators: Engagement and reward clarity (Q8–Q9)

Linked Figures: See Appendix A, Figure A8 and Figure A9

Outcomes:

- Both Q8 (engagement) and Q9 (clarity of points) received 100% Yes responses.
- Figures A8 and A9 show that all six users found the quiz enjoyable and understood the reward system.
- Some feedback noted that while the quiz was fun, users wanted the ability to see correct answers and revisit incorrect responses.

Feedback Sample:

“The quiz is fun, but I wish I could see the right answer after I get it wrong.”

5.2.6 Daily Login Streak Tracking

Indicators: Motivation to return (Q10), clarity and appeal of UI (Q11)

Linked Figures: See Appendix A, Figure A10 and Figure A11

Outcomes:

- Q10 and Q11 both received 100% Yes responses, showing that the streak system was motivational and well-designed.

- Figures A10 and A11 highlight consistent appreciation for the 🔥 icon and the perceived reward effect of streaks.
- Several users suggested linking streaks to actual bonus points or badges to further increase value.

Feedback Sample:

“The fire icon is cool. Makes me want to keep going and not break the chain.”

5.2.7 Overall Feature Satisfaction

Indicators: User sentiment across all features (Q12)

Linked Figure: See Appendix A, Figure A12

Outcomes:

- Q12 showed an even spread of responses from 1 to 5 (Very Dissatisfied to Very Satisfied), with each score chosen by exactly one user.
- Figure A12 reflects this mixed satisfaction result, with no clear consensus.
- The overall average rating was 3.0/5, suggesting that while features worked technically, users desired more polish, onboarding, and content depth.

Feedback Sample:

“The app is cute and has many features, but still a bit rough in some places.”

5.3 User Engagement and Feature Impact

The user survey results offer meaningful insight into how the newly implemented TrekBuddy features influenced user interaction, motivation, and perceived usefulness during travel planning. While the sample size was limited to six participants, the feedback consistently highlighted high engagement levels with key functionalities—particularly real-time information and gamification elements.

Real-time weather emerged as one of the most practically useful features. As shown in Appendix 1, Figures A4 and A5, all six participants confirmed the weather display was both accurate and timely, and five out of six users (83.3%) found it useful for making activity decisions. This affirms the role of context-aware services in mobile travel apps, where environmental awareness (e.g., rain or temperature) can influence spontaneous travel behavior. One user shared:

“I saw it would rain later, so I changed plans to visit a museum instead of a park.”

— *Anonymous user feedback*

Similarly, restaurant recommendations were well-received. As highlighted in Figures A6 and A7, all participants agreed the recommendations were relevant to the selected city, and all requested more details such as reviews, ratings, or operating hours. This high engagement confirms user interest in localized and enriched content. Several users proposed a feature allowing travelers to write and view reviews—now conceptually titled Rakho. These suggestions indicate a demand for socially-driven decision-making features in future versions of the app.

Gamification elements, including the quiz feature and daily login streaks, were particularly effective in maintaining engagement. As seen in Figures A8 through A11, all six users found the quiz fun and clear in how points were awarded. All users also reported that the streak feature motivated them to open the app more frequently. The 🔥 streak display was described as "visually rewarding" and encouraged consistent use.

“The fire icon is cool. Makes me want to keep going and not break the chain.”

— *Anonymous user feedback*

However, users also recommended enhancements to the quiz system to increase its educational and interactive value. Specifically, they wanted to review correct answers after submitting and suggested tailoring questions by location. This reflects a desire for both entertainment and learning value within the same feature.

“The quiz is fun, but I wish I could see the right answer after I get it wrong.”

Despite the positive response to individual features, overall satisfaction (Figure A12) was more mixed. Ratings were evenly distributed from 1 (Very Satisfied) to 5 (Very Dissatisfied), resulting in an average satisfaction score of 3.0/5. This suggests that while the core functions worked as intended, some users felt the app lacked polish or additional depth.

In addition, several users mentioned UI inconsistencies or bugs (e.g., "go back" button not working), and others expressed interest in features like social login, onboarding guides, and editable profiles—elements that support a smoother long-term user experience.

In summary, the survey results show that user engagement was driven primarily by three elements:

- Real-time contextual tools (weather, restaurants)
- Light gamification (quiz, streak)
- Desire for personalization and richer content (user reviews, tailored quizzes)

Although early-stage satisfaction was varied, the feedback reflects meaningful engagement and strong potential for further development. TrekBuddy's foundation successfully supports user needs in travel planning, and upcoming iterations—particularly with the addition of user reviews, more content-driven quizzes, and technical refinements—will likely increase both retention and satisfaction.

6 Discussion

6.1 New Feature Evaluation

The TrekBuddy mobile application has been enhanced through the implementation of four key features: multilingual support, real-time weather display, restaurant recommendations, and a quiz-based gamification system with daily streak tracking. These additions align with the thesis objective of improving the application's functionality and user experience while advancing technical expertise in React Native, Firebase, and API integration. By addressing user needs for accessibility, contextual information, and engagement, these features contribute to the project's goal of developing a modern, intuitive travel companion. Simultaneously, their development has supported personal learning outcomes, particularly in cross-platform development, modular UI design, and scalable feature implementation within the Expo ecosystem.

Each feature directly supports the project’s objectives of enhancing user experience and technical proficiency. The multilingual support, implemented using i18n, enables seamless language switching between English and Vietnamese, promoting inclusivity and accessibility for a broader audience. This required understanding of state management and asynchronous language loading, contributing to practical skills in internationalization. Real-time weather integration, powered by the OpenWeatherMap API, provides current conditions for searched cities, offering practical trip-planning support. This task enhanced the ability to manage asynchronous API calls and display dynamic data in a mobile environment. The restaurant recommendation feature, leveraging the Google Places API, delivers dynamic dining suggestions based on location, enriching the app’s utility beyond tourist destinations. Working with this task helps reinforce knowledge of external API integration and location services. The quiz game system introduces gamification to encourage daily interaction, aligning with the goal of increasing user retention. These features reflect successful integration of external APIs and Firebase, demonstrating technical growth in state management, internationalization, and real-time data handling.

The strengths of these features lie in their robust technical implementation and user-centric design. The use of external APIs and Firebase ensures reliable data retrieval and storage, seamlessly integrated into the React Native front-end. Multilingual support enhances usability for diverse users, while the weather and restaurant features provide immediate, practical value, reducing the need for external apps. The gamification system fosters engagement through progress tracking and rewards, supporting long-term user interaction. However, limitations exist: multilingual support is currently limited to two languages, with some translations requiring refinement. The weather display lacks extended forecasts or alerts, and restaurant recommendations omit ratings or filters,

potentially limiting decision-making utility. The quiz feature's static question set and basic streak display (text-only, without additional incentives) may reduce long-term replayability.

In conclusion, the new features significantly enhance TrekBuddy's value as a travel companion app, fulfilling the thesis goals of improving usability and demonstrating technical maturity. The learning outcomes include mastery of key mobile development skills, such as API integration, internationalization, state management, and gamified user interaction. Future improvements—such as adding more languages, integrating richer weather and location data, and expanding gamification elements with dynamic rewards—would further advance both the application's utility and the developer's skillset. These efforts collectively reflect a comprehensive and evolving understanding of scalable, user-focused mobile application development.

6.2 Key Learning Outcomes and Challenges

The development of TrekBuddy was a product-based project driven by the goal of creating a functional, multilingual mobile travel application with real-time weather, location-based restaurant suggestions, and gamified engagement. This section reflects on the key technical and design learnings gained throughout the project, as well as the challenges encountered during the iterative development process. Insights are drawn from user feedback (Appendix 1), implementation milestones, and the constraints of testing within the Expo development environment.

6.2.1 Key Learning Outcomes

This thesis project substantially supported my personal growth in cross-platform mobile development, particularly through the application of Firebase Firestore within a new development environment—React Native with JavaScript and Expo Application Services (EAS). Having previously worked with Firebase in a Flutter/Dart-based project, this work expanded my capabilities by exposing me to the React ecosystem, where integration patterns and architectural decisions differ significantly.

Working with Firebase Firestore, I gained hands-on experience in real-time database operations such as reading, writing, and updating structured user data across multiple features including quiz points, credits, login streaks, and profile metadata. This is demonstrated by Firestore database fields accurately reflect the users' data, and the app successfully updates the users' information (Figure 48-49).

The integration of third-party APIs—namely, OpenWeather and Google Places—developed my technical skills in asynchronous API handling, environment variable configuration, and location-based data rendering, all of which are essential for real-world travel applications. This aligned with

my objective to gain confidence in robust API integration for dynamic, real-time features. This is demonstrated by real-time weather successfully displayed on the app when user chooses the city (Figure 32-35)

Building multilingual support using `react-i18next` provided valuable insights into internationalization (`i18n`) and the importance of localized UI/UX. The system was successfully deployed to support English and Vietnamese, and user feedback confirmed that language switching was smooth and accurate (Section 5.2.2 and Section 5.2.3, Q2–Q3: 100% success rate).

The quiz-based gamification system—including a point reward structure and daily login streaks—was an essential part of learning how to maintain state across sessions using `AsyncStorage` and `Firebase Firestore`. This helped reinforce the value of persistent state management and visual feedback (e.g., streak icons) in maintaining user engagement (Section 5.2.5 and Section 5.2.6, Q8-Q9: 100% success rate).

Feature implementation was organized using a task-based Github Project board with checklist-style sprint items. This Agile-inspired workflow allowed rapid iteration and integration of feedback from a closed user test. This included breaking down features into sprints, incorporating user feedback into the backlog, and tracking progress across development milestones. It helped cultivate a product-oriented mindset focused on usability, iteration, and long-term maintainability. For example, several users requested more details in restaurant listings and quiz feedback mechanisms. This led to an evolving roadmap prioritizing real-world usability over purely technical novelty.

One major area of learning came from designing scalable code structures using custom hooks, modular components, and shared context providers for things like language, theme, and user session state. This structure greatly improved reusability and simplified the process of integrating new features like weather and quiz results without impacting other screens.

The project also introduced me to Expo and EAS, tools that were new to me at the beginning. I learned how to configure builds, manage deployment channels, and handle asset bundling across platforms. Although Expo Go facilitated fast iteration, I also encountered its limitations—particularly around native feature testing and automation. This highlighted the importance of adopting tools like Jest or Detox for more thorough and scalable test coverage in future work.

6.2.2 Challenges

While specific challenges related to each feature were discussed in detail in Chapter 4, this section highlights additional overarching challenges encountered during the overall development of the

TrekBuddy application, including technical limitations, testing constraints, and project management hurdles.

Expo Go limitations and manual testing gaps

Since all testing was done manually through Expo Go, it was difficult to simulate certain user conditions (e.g., offline mode, background refresh). Without automated testing, it was not feasible to consistently check all edge cases. For example, a bug in the saved screen's "go back" functionality was discovered late due to inconsistent manual testing (Appendix 1, Figure A13). This highlighted the need for automated unit and integration testing in future development.

Implementing gamification logic and streak tracking

Creating a consistent daily streak system required handling time zones, tracking login history, and updating streak icons. Time comparison logic had to be carefully implemented to ensure streak continuity even across different time zones or after 23:59 activity, which proved surprisingly complex.

Firebase Firestore query structuring and security

Managing multiple collections (e.g., user data, quizzes, streaks, city selections) required precise querying and secure rule configuration. Structuring Firestore to support efficient reads while minimizing billing was a significant learning experience.

Multilingual translation quality

Although the i18n engine worked well technically, a few translated Vietnamese strings lacked context-specific phrasing. Localizing idiomatic or UI-specific phrases accurately requires human review and ongoing refinement.

Feature creep vs. performance

Adding new ideas like user review system or expanded quiz categories posed architectural trade-offs. Balancing ambition with mobile performance constraints required difficult decisions about what to prioritize within a tight development timeline.

UI responsiveness across devices

Creating a consistent look across Android and iOS screens required constant testing and tweaking of styles, especially when using react-native layout constraints and third-party components. Button alignment, font scaling, and modal responsiveness often needed repeated tuning.

Time management and scope balancing

As a solo developer, it was challenging to both implement features and gather feedback while maintaining code quality. Some planned features, like review submissions and advanced filtering, were deferred due to time constraints, highlighting the importance of iterative release planning.

6.3 Future Improvements and Enhancements

TrekBuddy offers a strong foundation with a rich feature set and user-friendly interface, but the user survey highlighted several areas for improvement to further enhance functionality, engagement, and overall experience. Future development should prioritize expanding app capabilities, improving usability based on user feedback, and ensuring technical robustness for scalable deployment.

Survey respondents emphasized the importance of richer restaurant information, with 100% requesting additional details such as opening hours, ratings, and user-generated reviews. To address this, a user review and rating system should be developed, allowing travelers to leave feedback and share recommendations. This would not only improve content credibility but also support community-driven engagement.

Gamification features like the quiz and daily streak were well-received; however, users suggested improvements such as showing the correct answer after each quiz and tailoring questions based on the selected destination. Expanding quiz categories to include travel trivia, culture, and local history could also increase educational value and enjoyment.

Several users also noted that the app lacked clear guidance during initial use. To improve onboarding, a first-time user tutorial, tooltip walkthrough, or interactive guide should be implemented to clarify feature functions and navigation. In addition, while language switching performed well, refining some Vietnamese translations and expanding to additional languages (e.g., Finnish, Chinese) would enhance accessibility for a broader audience.

Improving inclusivity and accessibility is another critical area. TrekBuddy should support dynamic text resizing, high-contrast themes, and screen reader compatibility to serve users with visual or cognitive impairments.

From a development standpoint, one technical limitation currently lies in the testing process. The app has been tested manually using Expo Go, which provides rapid prototyping but lacks full access to native debugging tools and automation. As a result, certain bugs or performance issues may go undetected in manual testing alone. Therefore, a future goal is to implement automated

testing using tools like Jest, Detox, or Cypress to validate core functionalities, ensure reliability across devices, and maintain development scalability as the codebase grows.

To summarize, key improvement areas for future development include:

- Implementing user reviews and ratings for restaurants and attractions
- Enhancing the quiz system with correct answers and location-based content
- Developing a first-time onboarding guide for clearer navigation
- Refining translations and adding more language options
- Expanding accessibility features (dynamic font scaling, screen readers, contrast modes)
- Transitioning to automated testing to improve code quality and reduce manual errors

These improvements, grounded in user feedback and technical best practices, will help TrekBuddy evolve into a more comprehensive, accessible, and reliable travel companion for global users.

7 Conclusion

7.1 Summary of Findings

In conclusion, the development of the TrekBuddy mobile application successfully fulfilled its core objectives: enhancing travel-related functionality, improving user experience, and supporting personal technical growth. Beginning with essential features such as city search, destination exploration, and map integration, the app laid a strong foundation as a user-friendly travel companion. Subsequent additions—real-time weather updates, bilingual language support, restaurant recommendations, and a gamified quiz system—transformed the app into a more engaging, informative, and accessible tool for travelers.

From a technical perspective, the project provided valuable hands-on experience in modern mobile development using React Native, Firebase, and various third-party APIs. Skills developed throughout the process included real-time data management, secure user authentication, scalable architecture design, and cross-platform deployment using Expo and EAS. The successful integration of these tools demonstrates the feasibility of building complex, responsive mobile applications independently, and highlights best practices in modular UI design, state management, and internationalization.

Overall, TrekBuddy evolved from a simple concept into a feature-rich, scalable mobile application with real-world utility. The project not only met its functional and educational goals but also laid a foundation for future enhancements and expansion. With further development—such as support for offline access, richer content, and broader language options—TrekBuddy has the potential to become a fully comprehensive travel assistant, and the knowledge gained from its development offers long-term value for future software projects.

7.2 Contributions of the Thesis

This thesis presents both theoretical and practical insights into mobile application development, especially concerning individual development projects. From an academic standpoint, it offers a detailed case study illustrating how contemporary technologies like React Native, Firebase, and various APIs can be integrated to create a cross-platform mobile application. The thesis showcases the cyclical process of design, development, and testing while providing important perspectives on agile development methodologies, user interface/user experience considerations, and strategies for optimizing mobile applications.

The TrekBuddy project essentially acts as a detailed resource for developers interested in creating real-world applications with the React Native framework. Its documentation, architecture, and

integration of services like Google Maps and the OpenWeather API provide clear, step-by-step examples and reusable methodologies. It addresses practical solutions to frequent issues, such as managing state, navigating workflows, ensuring data persistence, and supporting multiple languages. These examples can be particularly beneficial for students, hobbyists, or junior developers looking to enhance their abilities in independently building and deploying mobile applications.

Sources

Adomavicius, G., & Tuzhilin, A. 2011. Context-aware recommender systems. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), Recommender systems handbook (pp. 217–253). Springer. Link: https://doi.org/10.1007/978-0-387-85820-3_7. Accessed on 27/04/2025.

Bhagavatiprasad Vaghela. February 2023. The Benefits of Using React Native for Mobile Development. Link: https://www.computer.org/publications/tech-news/trends/benefits-of-react-native. Accessed on 10/03/2025.

Bhagya. November 2021. Introduction to Firebase. Link: https://bki17.medium.com/introduction-to-firebase-61d391a3c443. Accessed on 13/03/2025.

Cyber Yodha. March 2023. What is Firebase. Link: https://www.cyberyodha.org/2023/03/what-is-firebase.html. Accessed on 14/03/2025.

DailyHostNews. February 2019. Google's NoSQL database service Cloud Firestore now up for grabs. Link: https://www.dailyhostnews.com/googles-nosql-database-service-cloud-firestore-now-up-for-grabs. Accessed on 14/03/2025.

Deci, E. L., & Ryan, R. M. 2000. The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior. Psychological Inquiry, 11(4), 227–268. Link: https://doi.org/10.1207/S15327965PLI1104_01. Accessed on 27/04/2025.

Deci, E. L., & Ryan, R. M. 2000. The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior. Psychological Inquiry, 11(4), 227–268. Link: https://doi.org/10.1207/S15327965PLI1104_01. Accessed on 26/04/2025.

Deterding, S., Dixon, D., Khaled, R., & Nacke, L. 2011. From game design elements to gamefulness: Defining "gamification". In Proceedings of the 15th International Academic MindTrek Conference (pp. 9–15). ACM. Link: https://doi.org/10.1145/2181037.2181040. Accessed on 01/05/2025.

Eisenman, B. 2021. Learning React Native, 2nd Edition. O'Reilly Media. E-book. Accessed on 18/03/2025.

Endsley, M. R. 1995. Toward a theory of situation awareness in dynamic systems. Human Factors, 37(1), 32–64. Link: https://doi.org/10.1518/001872095779049543. Accessed on 22/04/2025.

Expo. (n.d.). Expo Documentation. Link: https://docs.expo.dev/. Accessed on 15/03/2025.

Expo EAS. (n.d.). EAS Build. Link: https://docs.expo.dev/build/introduction/. Accessed on 20/04/2025.

Geeksforgeeks. February 2025. Firebase – Introduction. Link: https://www.geeksforgeeks.org/firebase-introduction/. Accessed on 10/03/2025.

Geeksforgeeks. March 2024. Getting started with React Native. Link: https://www.geeksforgeeks.org/getting-started-with-react-native-read-this-first/#what-is-react-native-. Accessed on 09/03/2025.

Google. (n.d.). Firebase. Link: https://firebase.google.com/. Accessed on 10/03/2025.

Google Cloud. (n.d.a). Places API. Link: https://developers.google.com/maps/documentation/places/web-service/overview. Accessed on 18/03/2025.

Google Cloud. (n.d.b). Geocoding API. Link: https://developers.google.com/maps/documentation/geocoding/overview. Accessed on 18/03/2025.

Google Cloud. (n.d.c). Routes API. Link: https://developers.google.com/maps/documentation/routes. Accesed om 19/03/2025.

Google Cloud. (n.d.d). Address Validation API. Link: https://developers.google.com/maps/documentation/address-validation. Accessed on 20/04/2025.

Google Maps Platform (n.d). Google Places API. Link: https://developers.google.com/maps/documentation/places/web-service. Accessed on 10/03/2025.

Harrington, J. L. 2019. Cloud computing for small businesses: A practical guide to scalability and efficiency. Apress. E-book. Accessed on 23/03/2025.

Hassenzahl, M. 2005. The thing and I: Understanding the relationship between user and product. In M. A. Blythe, K. Overbeeke, A. F. Monk, & P. C. Wright (Eds.), Funology: From usability to enjoyment (pp. 31–42). Springer. Link: https://doi.org/10.1007/1-4020-2967-5_4. Accessed on 23/03/2025.

Hassenzahl, M. 2010. Experience design: Technology for all the right reasons. Synthesis Lectures on Human-Centered Informatics, 3(1), 1–95. Link: https://doi.org/10.2200/S00261ED1V01Y201003HCI008. Accessed on 31/03/2025.

i18next documentation. (n.d.). Getting started. Link: https://www.i18next.com/overview/getting-started. Accessed on 18/04/2025.

Invertase Limited. 2025. React Native Firebase. Link: https://rnfirebase.io/. Accessed on 10/03/2025.

Krug, S. 2014. Don't Make Me Think!: A Common Sense Approach to Web Usability, Second Edition. New Riders. E-book. Accessed on 13/03/2025.

Marcus, A., & Gould, E. W. 2000. Crosscurrents: Cultural dimensions and global Web user-interface design. Interactions, 7(4), 32–46. Link: https://doi.org/10.1145/345190.345238. Accessed on 24/04/2025.

Material-UI (MUI) (n.d.). Material-UI Documentation. Link: https://mui.com/getting-started/. Accessed on 20/03/2025.

Meta Open Source. 2025. React Native. Link: https://reactnative.dev/. Accessed on 10/03/2025.

Nguyen Van Tam. September 2021. Giải đáp các câu hỏi thường gặp với Firebase Cloud Messaging Service. Link: https://viblo.asia/p/giai-dap-cac-cau-hoi-thuong-gap-voi-firebase-cloud-messaging-service-eW65G6jjlDO. Accessed on 14/03/2025.

Piyush Nanwani. January 2024. React Native CLI vs Expo CLI – Comparison. Link: https://blog.atomxel.com/react-native-cli-vs-expo-cli-comparison. Accessed on 13/03/2025.

React Native Google Places Autocomplete. (n.d.). React Native Google Places Autocomplete documentation. Link: https://www.npmjs.com/package/react-native-google-places-autocomplete. Accessed on 20/03/2025.

Software Mansion. 2025. React Native Reanimated Getting Started. Link: https://docs.swmansion.com/react-native-reanimated/docs/fundamentals/getting-started/. Accessed on 04/04/2025.

React Native. (n.d.). React Native: Learn once, write anywhere. Link: https://reactnative.dev/docs/getting-started. Accessed on 15/03/2025.

React Navigation. (n.d.). React Navigation: Routing and navigation for your React Native apps. Link: https://reactnavigation.org/docs/getting-started/. Accessed on 16/03/2025.

Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data and repealing Directive 95/46/EC (General Data Protection Regulation).

Sakhniuk, M., & Boduch, A. 2020. React and React Native - Fifth Edition. Packt Publishing. Ebook. Accessed on 17/03/2025.

Sadia Aziz. 2025. 7 Key Benefits of Choosing React Native for Your Next Project. Link: https://invozone.com/blog/key-benefits-of-choosing-react-native-for-your-next-project/. Accessed on 13/03/2025.

Schwartz, D. G., Te'eni, D., & Markus, M. L. 1999. The role of language in knowledge management: A review of the literature. Journal of Information Science, 25(5), 411–424. Link: https://doi.org/10.1177/016555159902500505. Accessed on 25/04/2025.

Shneiderman, B., & Plaisant, C. 2010. Designing the user interface: Strategies for effective human-computer interaction (5th ed.). Pearson. E-book. Accessed on 22/03/2025.

Singh, G., & Wesson, J. L. 2009. Evaluating the usability of a user interface for a mobile application. In Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists (pp. 251–257). ACM. Link: https://doi.org/10.1145/1632149.1632182. Accessed on 15/04/2025.

Skinner, B. F. 1953. Science and human behavior. New York: Macmillan. Accessed on 28/04/2025.

Software Mansion. (n.d.). React Native Gesture Handler. Link: https://docs.swmansion.com/react-native-gesture-handler/docs/. Accessed on 17/03/2025.

Stallings, W. 2017. Cryptography and network security: Principles and practice (7th ed.). Pearson. E-book. Accessed on 19/03/2025.

Vladimir Novick. 2017. React Native - Building Mobile Apps with JavaScript. O'Reilly Media. E-book. Accessed on 24/03/2025.

Appendices

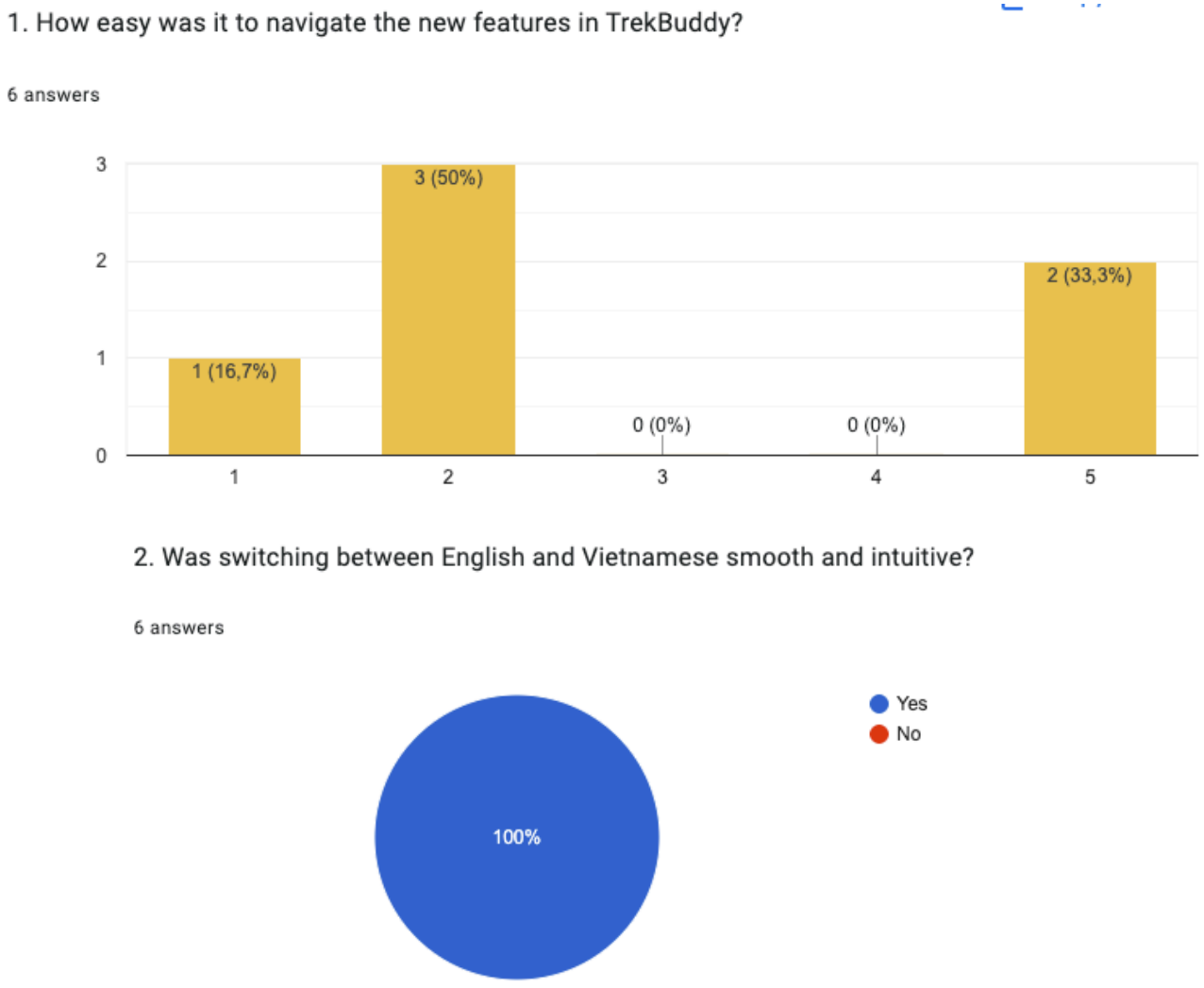
Appendix 1. Survey Results – Visual Summary

This appendix provides the complete set of visual survey results referenced in Chapter 5. Each figure corresponds to an individual survey question (Q1–Q13) and illustrates user responses through bar charts or pie charts, capturing key trends in satisfaction, usability, feature engagement, and improvement suggestions. These visualizations support the quantitative analysis and offer a clearer understanding of user feedback and overall experience with the TrekBuddy application.

Figure	Description	Question
A1	User responses on ease of navigating new TrekBuddy features (bar chart)	Q1
A2	Pie chart showing users' experience switching between English and Vietnamese	Q2
A3	Pie chart indicating whether users encountered translation issues	Q3
A4	Pie chart showing if weather data was displayed quickly and accurately	Q4
A5	Pie chart showing if weather information was useful for travel planning	Q5
A6	Pie chart showing relevance of recommended restaurants	Q6
A7	Pie chart showing interest in additional restaurant details	Q7
A8	Pie chart showing quiz engagement and enjoyment	Q8
A9	Pie chart showing clarity of point/credit rewards after quizzes	Q9
A10	Pie chart showing if the login streak feature motivated frequent app use	Q10
A11	Pie chart showing clarity and visual appeal of the streak display	Q11
A12	Bar chart showing overall satisfaction with new features	Q12
A13	List of user suggestions and feature improvement ideas	Q13

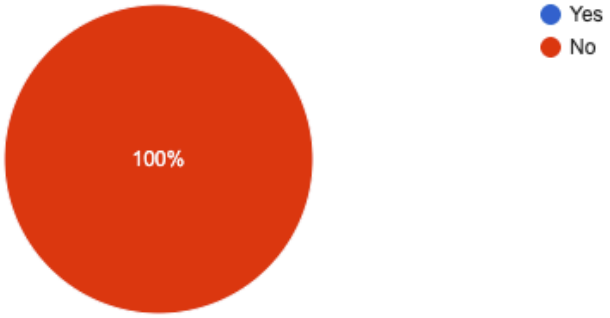
A14	Survey question using a Likert scale to rate how easy it was to navigate TrekBuddy's new features.	
A15	Survey question using a Likert scale to rate how satisfied it was with TrekBuddy's new features.	

Table 3. Survey figures with description and according questions.



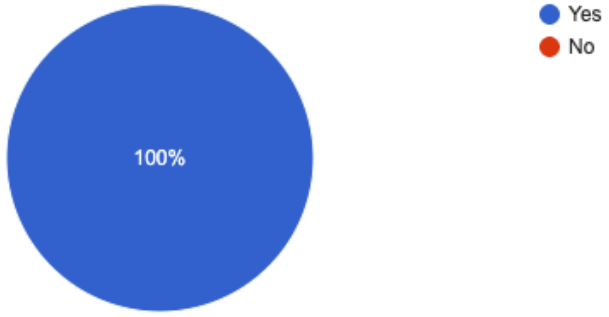
3. Did you experience any translation issues or unclear phrases?

6 answers



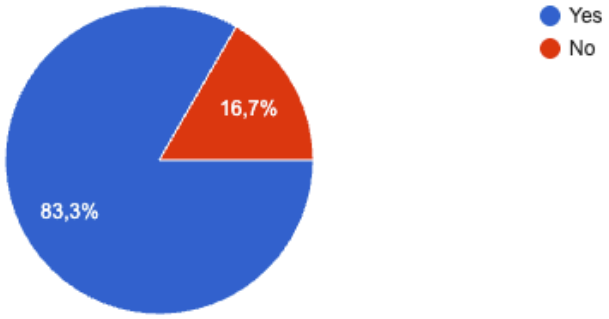
4. Was the weather information displayed quickly and accurately when viewing a city?

6 answers



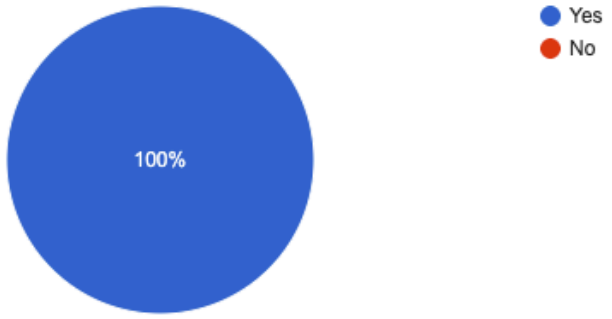
5. Was the weather information useful for your travel planning?

6 answers



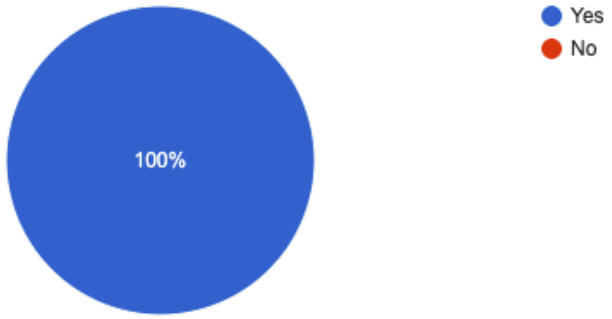
6. Did you find the recommended restaurants relevant to your selected city?

6 answers



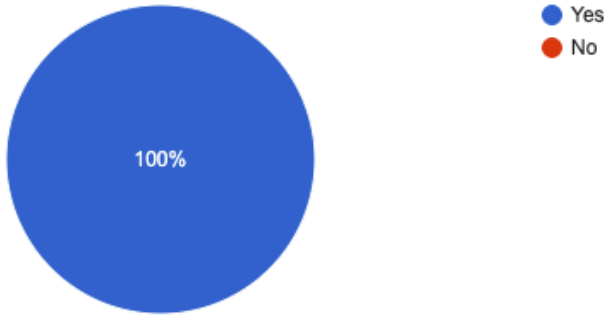
7. Would you like to see additional restaurant details (e.g., ratings, hours)?

6 answers



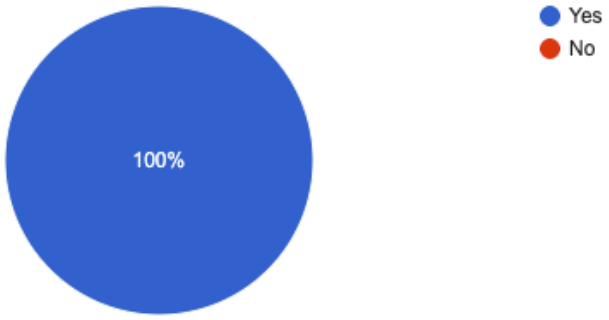
8. Was the quiz feature engaging and enjoyable to play?

6 answers



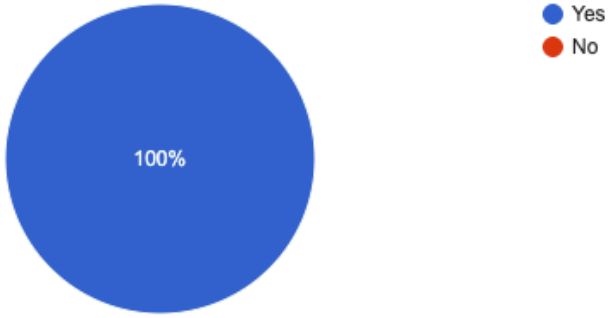
9. Was it clear how points or credits are awarded after each quiz?

6 answers



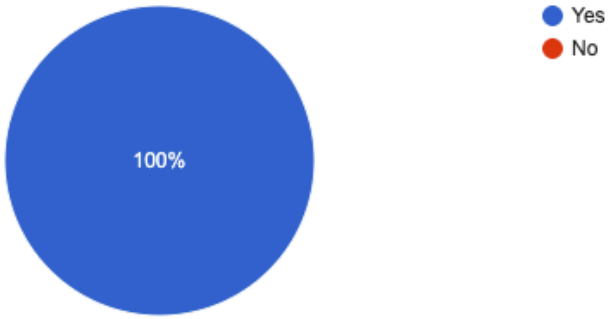
10. Did the daily login streak feature motivate you to open the app more often?

6 answers



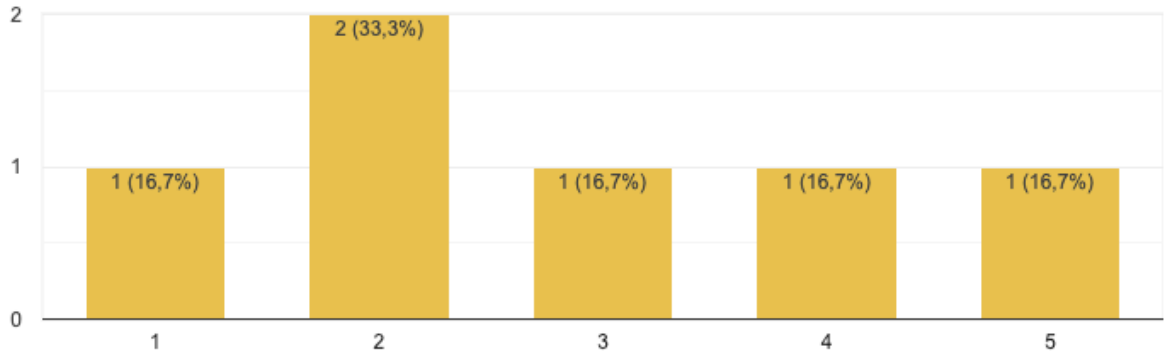
11. Was the streak display (e.g., 🔥 3-day streak) clear and rewarding?

6 answers



12. How satisfied are you with the new features overall?

6 answers



13. What improvements or additional features would you like to see in the future?

6 answers

Quiz game is fun. But would be better if the quiz is about travel/places/city etc since its the app's main content
Review/likes for restaurant is appreciated. So the most favorite will be displayed on top

Perhaps quiz tailored to the specific city. And other users can write comments about the specific restaurants/cities.

I want to see the ratings and feedback from customers
For the game and quiz section, it would be nice if I can get to know the correct answer after I answered it wrong or if I can choose the answer again if I was wrong

The restaurant rating, review and the map to go to there after saving the city and restaurant

sign in with other account (google, facebook etc...). Change profile info. More language options. Maybe a quick guide on how to use the app when new user sign in the first time.

After I did the quiz, the profile did not show the points. The "go back" button in the saved screen isn't working yet. The app is cute and has many features.

How easy was it to navigate the new features in TrekBuddy? *

	1	2	3	4	5	
Very Easy	○	○	○	○	○	Very Difficult

How satisfied are you with the new features overall? *

	1	2	3	4	5	
Very Satisfied	○	○	○	○	○	Very Dissatisfied

Appendix 2. Early User Interviews (Pre-Implementation Research)

In January 2025, four informal user interviews were conducted with individuals aged 20–35 who frequently use mobile apps for travel planning. Each interview focused on identifying travel-related needs, pain points, and app usage behavior. Participants were asked open-ended questions about what they find useful or lacking in existing travel apps.

Interview Format

Informal one-on-one conversations with duration is 5–10 minutes each. The interview was done in-person and online chat. Participants for this interview are 2 Vietnamese male, 2 Finnish female with aged 22–34.

Q#	Questions
Q1	What travel-related apps do you currently use?
Q2	What features do you often rely on while traveling?
Q3	Have you used apps with quizzes or streaks before?
Q4	Would switching between English and Vietnamese be useful for you or your family?

Table 4. Questions for an informal interview about user needs for future features in travel application.

Below are the key insights and representative feedback that were collected after the informal interview:

“I want to know where to eat without having to open another app like Google Maps.”

“Sometimes I forget to check the weather. If an app showed it directly, it would help a lot.”

“My mom speaks Vietnamese. I speak English. It’s better if both can use the same app.”

“Apps like Duolingo make me want to open them daily. I’d love something like that in travel.”

“A short quiz about the place I’m visiting would be fun—especially if it gave me points.”

Appendix 3. Market Review (Pre-Implementation Research)

To evaluate how existing applications addressed similar needs, a focused comparison was made between Google Travel and TripAdvisor, two widely used travel planning platforms. The aim was to identify commonly implemented features and potential gaps that TrekBuddy could fill.

Below is the feature comparison table among Google Travel, TripAdvisor, and TrekBuddy.

Feature	Google Travel	TripAdvisor	TrekBuddy Objective
Real-time weather display	No	No	Yes
Restaurant recommendations	No	Yes	Yes
Multilingual user interface	No	Yes	Yes (English–Vietnamese toggle)
Gamified quiz or streak system	No	No	Yes (Quiz + login streak)

Table 5. Feature comparison among travel applications

Neither app includes real-time weather integration or gamification, suggesting room for differentiation. TripAdvisor offers multilingual support and reviews, but lacks playful or interactive engagement tools. Google Travel is strong on logistical planning but does not provide language flexibility or user retention features like streaks or quizzes.

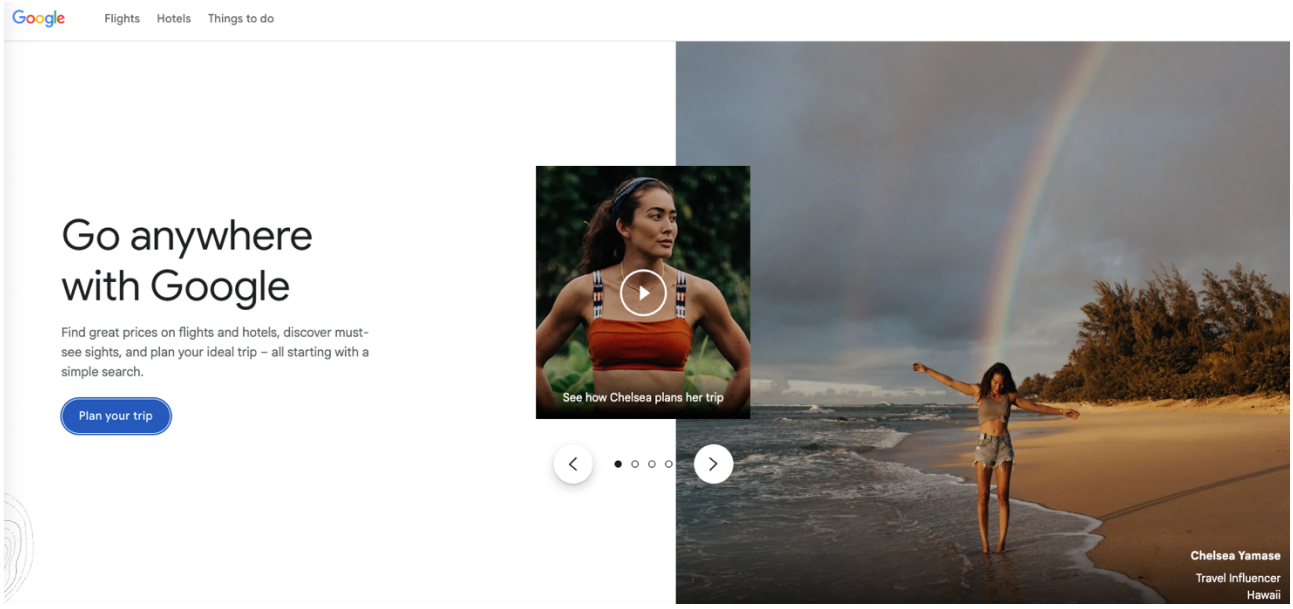


Figure C1. Google Travel platform.

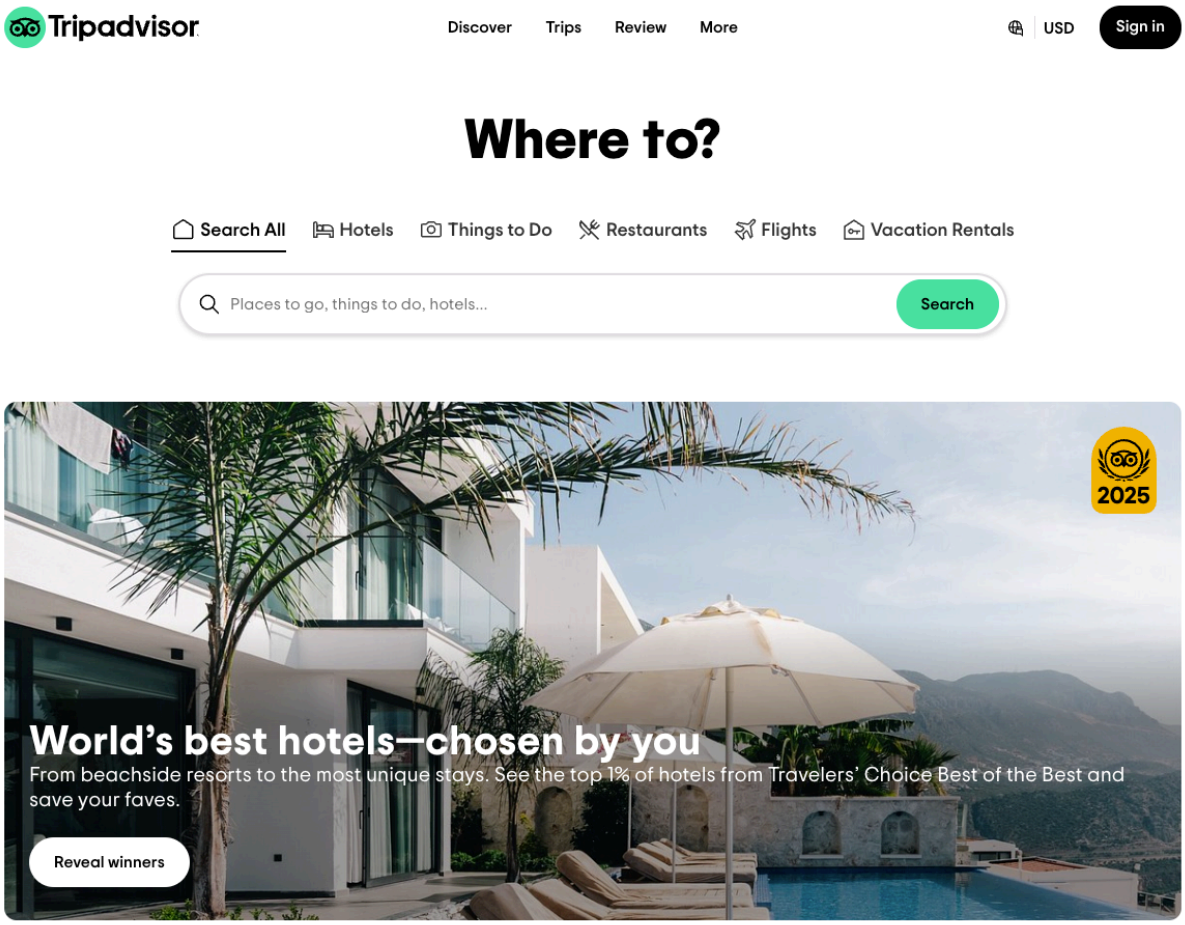


Figure C2. TripAdvisor platform.

Appendix 4. Github Repository

The source code for the mobile travel application – TrekBuddy – developed in this thesis can be found at:

https://github.com/anhng1106/TrekBuddy_mobile_app_project