Software Requirements Specification

for

TransitEase

Version 3.0 approved

Version 3 approved

Prepared by Ashwin, Dave, Jun Heng, Jonathan

Nanyang Technological University

10/11/24

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Ashwin	10/11	Updated External Interface Requirement	3.0
Dave	10/11	Updated Introduction and Overall Description Section	3.0
Jonathan	10/11	Updated System Feature Section	3.0
Goh Jun Heng	10/11	Updated the Requirement Section	3.0

1. Introduction

1.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for the **TransitEase** app, which is currently in version 3.0. This document will serve as a reference for the development, testing, and deployment of TransitEase. It includes specifications for backend functionalities (data retrieval from the URA Carpark API), data processing and storage (using Firebase), and front-end interactions in a Flutter-based mobile application.

1.2 Document Conventions

Priority levels: Requirements are prioritized as **High**, **Medium**, or **Low** priority. High-priority requirements are critical for the app's basic functionality:

- **Bold Text**: Denotes essential terms and sections.
- Italicized Text: Indicates examples or explanations.

Requirement Numbering: Each requirement statement is uniquely numbered to facilitate easy tracking and reference during development and testing.
.

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers**: To understand the technical requirements and constraints of the app and back-end systems.
- **Project Managers**: To track the progress and scope of requirements and to ensure alignment with project timelines.
- **Testers**: To define test cases based on detailed functional requirements.
- **Users**: To get an overview of the intended features and functionalities (if applicable).
- **Documentation Writers**: To prepare user guides, FAQs, and other help materials.

1.4 Product Scope

The **TransitEase** app aims to provide a convenient and efficient way for users to check real-time car park availability, report issues, and customize user preferences for a personalized experience. The app primarily targets urban commuters who rely on public and private car parks, enabling them to make informed decisions on where to park based on current availability.

The app's backend handles data requests to the **URA Carpark API**, processes and updates car park data, and saves user preferences and feedback in **Firebase**. These capabilities align with corporate goals of enhancing urban mobility and providing digital solutions for convenience in everyday travel.

By offering reliable car park data and customization options, TransitEase aims to reduce traffic congestion, save users time, and improve urban infrastructure utilization.

1.5 References

1. **URA Carpark API Documentation:**

   Title : URA Data Service API Documentations
   Author : Urban Redevelopment Authority
   Version : 1.2
   Date : August 2024
   Location : https://www.ura.gov.sg/uraDataService

2. **Firebase Documentation**:

   Title: Firebase Authentication and Firestore Documentation
   Author: Google
   Version: Latest as of September 2024
   Location: https://firebase.google.com/docs

3. **Flutter Documentation**:

   Title: Flutter Development Documentation
   Author: Google
   Version: Latest as of September 2024
   Location: https://flutter.dev/docs

2. Overall Description

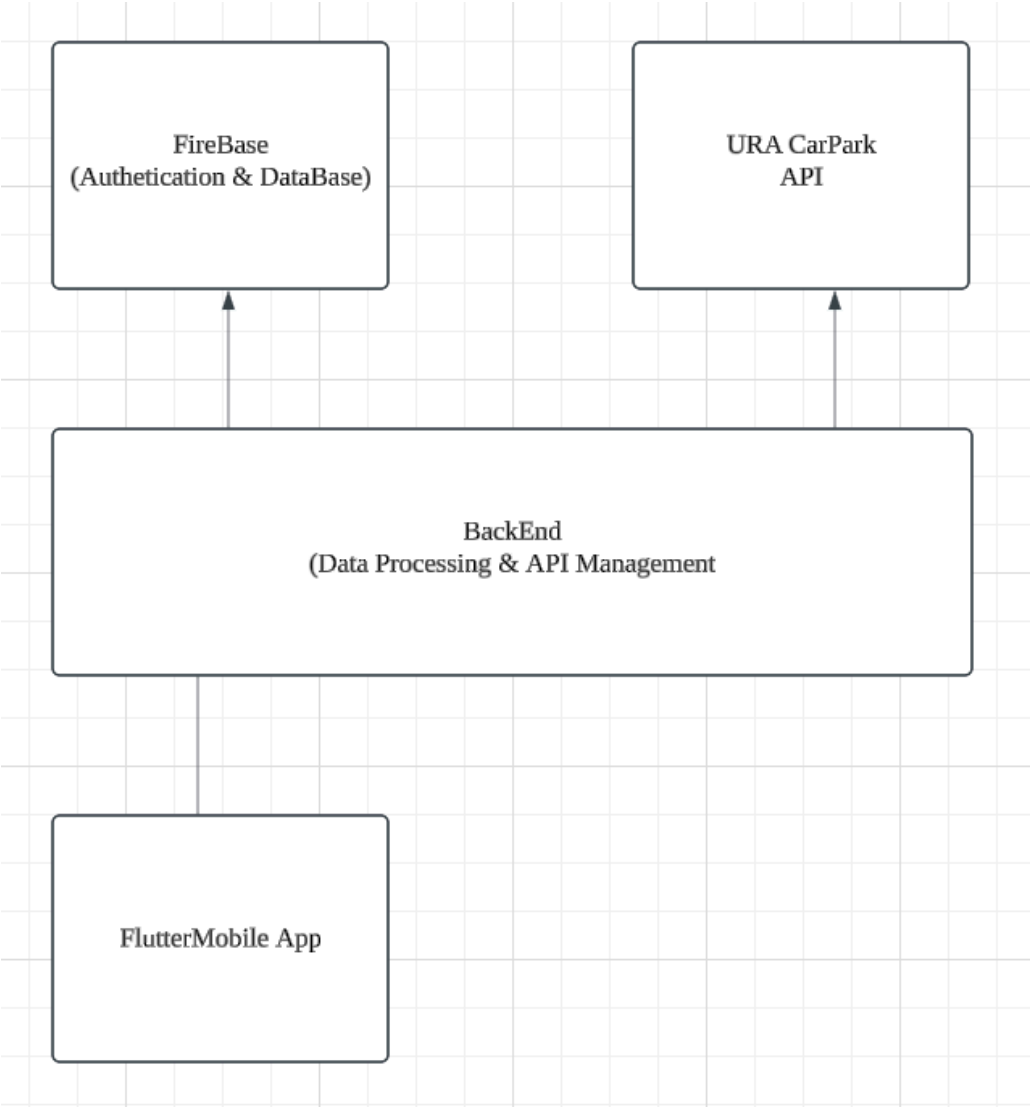
2.1 Product Perspective

The **TransitEase** app is a new, self-contained product designed to provide users with real-time car park availability data, preferences management, and feedback submission. It aims to improve urban mobility by assisting drivers in finding parking and optimizing parking infrastructure usage.

This product integrates several components:

- **URA Carpark API**: Provides real-time data on car park availability, allowing the backend to retrieve and process this information.
- **Firebase**: Used for user authentication, data storage, and preferences management.
- **Flutter-based Mobile App**: The user interface through which end-users interact with the system to check parking data, manage preferences, and report bugs or issues.

Below is a simple diagram illustrating the major components and interactions:



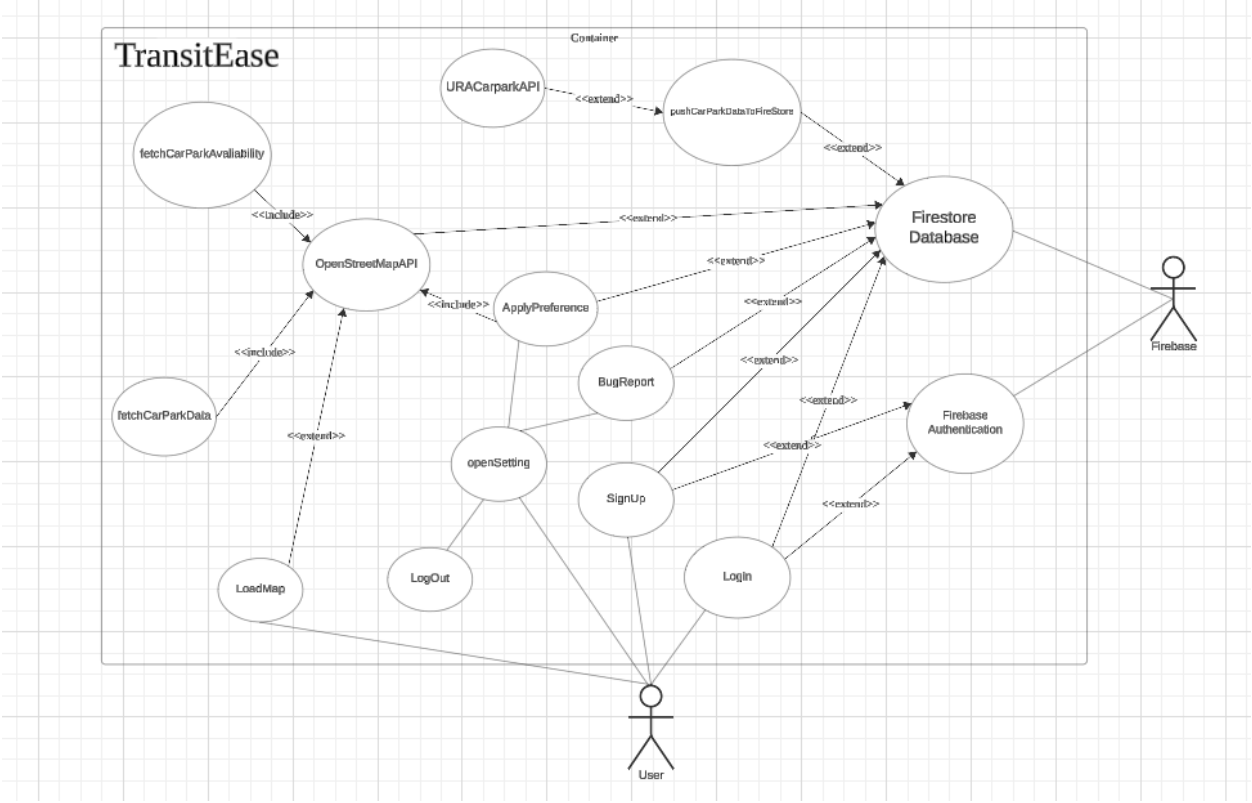
2.2 Product Functions

- The **TransitEase** app provides the following major functions:
- **User Authentication**:
  - Users can sign up
  - Log in using Firebase Authentication
  - Log out using Firebase Authentication.
- **Car Park Availability Check**:
  - Users can view real-time car park availability data.
- **Preferences Management**:
  - Users can set preferences
  - Update preferences for a customized experience.

    - Cloud saving for preferences.
- **Bug Reporting**:
    - Users can submit bug reports
    - Provide feedback about issues encountered within the app.
- **Data Processing**:
    - The backend retrieves data from the URA Carpark API,
    - Processes it and updates Firebase for the frontend to access.

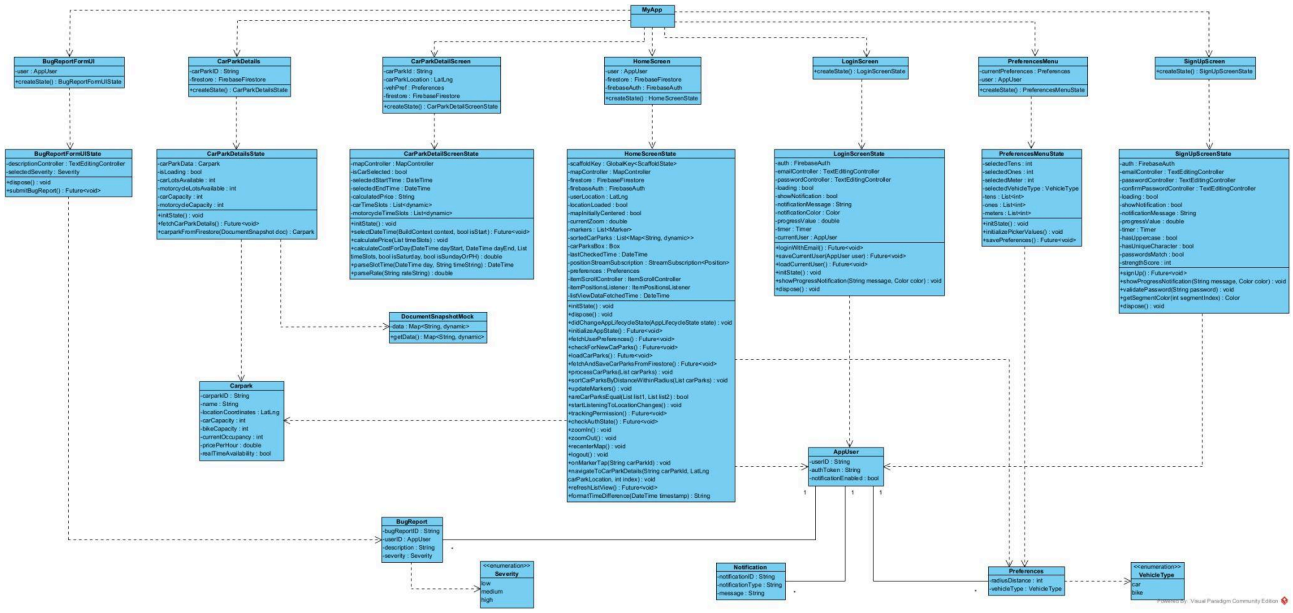
2.2.1 The UseCase Diagram

(shown below)
Depicts the interaction between the actors with the system itself with the inclusion of the functions mentioned above:



2.2.2 The following Class Diagram

(shown below)
Illustrates the interactions between various classes, along with the main functionalities required by TransitEase.



2.3 User Classes and Characteristics

The anticipated user classes for **TransitEase** are:

- **Users**: Typical drivers or urban commuters who use the app occasionally to check car park availability or manage preferences. They may not have technical expertise and require a user-friendly interface.
- **Administrators**: Technical team members responsible for maintaining the backend, managing Firebase data, and handling bug reports. Administrators have a higher level of technical expertise and access to more advanced monitoring tools.
- **Testers**: Responsible for testing the app's functionalities, including API integrations, data storage, and user interface consistency.

2.4 Operating Environment

- **Mobile Platform**: The app is developed for mobile devices and will run on **iOS (13+)** operating systems.
- **Backend**: Hosted in a cloud environment, the backend operates on **Docker** and interacts with the URA Carpark API.
- **Database**: Firebase Firestore, used to store user data, preferences, car park availability, and feedback submissions.
- **Other Technologies**: The app uses **Firebase Authentication** for user management, **Firebase Firestore** for data storage, and **Dart** as the primary programming language within the Flutter framework.

2.5 Design and Implementation Constraints

- **API Rate Limits**: The URA Carpark API has rate limits and may restrict the frequency of requests. This constraint affects how often car park data can be updated.

2.6 User Documentation

- **Video Demonstration**: A video demonstrating the main features of the app, including login, car park checks, preferences, and bug reporting.

- **Administrator Documentation**: Internal documentation for administrators and developers that includes deployment guides, API keys, and maintenance instructions on GitHub.

2.7 Assumptions and Dependencies

- **URA Carpark API**: Assumes that the URA Carpark API remains available and functional.
- **Firebase Services**: Assumes that Firebase Authentication and Firestore services are available and functional. Service outages could affect user login, data storage, and preference management.
- **Internet Connectivity**: Assumes that users have an active internet connection to retrieve real-time data and synchronize preferences.
- **Mobile Device Compatibility**: Assumes users are on supported iOS version for Flutter app to function properly.

3. External Interface Requirements

3.1 User Interfaces

The **TransitEase** app will have a user-friendly, intuitive interface designed according to standard mobile app conventions to ensure ease of use. Key characteristics of the user interface include:

Refer to Appendix D for Application Screenshots

- **Login and Registration Screens**: Allow users to sign up or log in using Firebase Authentication.
    - Each screen will include standard fields (e.g., email, password) and options for “Forgot Password” and “Create Account.”
    - Error messages (e.g., invalid password) will be displayed if login fails.
- **Main Dashboard**: Displays car park availability information on a map, updated in real time. The dashboard will include:
    - A map view showing the user’s location and nearby car parks with color-coded availability indicators.

    - A list view option displaying car park names, distance, and availability.

- **Preferences Screen**: Allows users to customize settings
    - such as preferred car park locations or notification settings.
- **Bug Report Screen**: Provides a simple form for users to report issues or provide feedback, including fields for a brief description and priority level.
- **Navigation Bar**: Includes icons for quick access to the dashboard, preferences, and bug reporting screens.
- **Standard Buttons and Functions**: Includes standard buttons like “Back,” “Home,” and “Help.” These buttons will appear at the top or bottom of each screen to ensure ease of navigation.
- **Error Messages and Notifications**: Clear error messages will be displayed to users for invalid actions, connectivity issues, or data fetching errors. Notifications will be used to alert users about new data or important updates.

3.2 Hardware Interfaces

- **Supported Devices**:
    - **iOS Devices**: iPhone models running iOS 13.0 or later.
- **GPS/Location Services**: The app will require access to the device’s GPS for location tracking to show the user’s position on the map and to calculate distances to nearby car parks.
- **Network Connectivity**: The app requires internet access (Wi-Fi or mobile data) to retrieve data from Firebase and the URA Carpark API.

3.3 Software Interfaces

The **TransitEase** app will interface with several external software components, each with specific roles:

Firebase:

- **Firebase Authentication**: Manages user login, registration, and logout. Uses standard authentication protocols to ensure secure access.
- **Firebase Firestore**: A cloud database for storing user preferences, car park data, and bug reports. Data in Firestore will be organized into collections (e.g., "car_parks" for car park data, "users" for preferences and settings).

URA Carpark API:

- **Data Retrieval**: The backend interacts with the URA Carpark API to fetch real-time car park availability data. Requests are authenticated using an access key and token, with data sent and received in JSON format.

- **Token Management**: The backend periodically retrieves a new token from the URA API to ensure uninterrupted data access.

Backend Processing:

- **Docker**: Used for containerizing the backend to ensure a consistent deployment environment.

Flutter Framework:

- **Dart Language**: Used for developing the TransitEase mobile app, utilizing Flutter widgets for cross-platform compatibility with IOS

3.4 Communications Interfaces

1. Overview of Communications Functions

The carpark application involves multiple communication layers and interactions between different components:

- **Flutter App**: Communicates with Firebase for data retrieval and updates, providing a seamless user experience.
- **Backend Services**: Dockerized Python services running scheduled tasks to process and update data in Firebase.
- **Firebase**: Acts as a cloud-based backend providing database, authentication, and data synchronization services.

2. Communications Functions

- **Real-time Data Synchronization**: Firebase provides real-time database functionalities, allowing the app to fetch car park data, user preferences, and updates on the fly.
- **Scheduled Data Processing**: The Docker container runs Python scripts as scheduled tasks to fetch, process, and push data updates to Firebase.
- **User Communication**: Firebase Authentication facilitates secure user login/logout, while Firestore serves as the database for storing user preferences and car park information.

3. Message Formatting and Communication Protocols

- **Message Formatting**:
  - **JSON**: Data exchanged between the Flutter app and Firebase is formatted as JSON objects for compatibility and easy parsing.
- **Communication Protocols**:
  - **HTTP/HTTPS**: Used by the Flutter app to communicate with Firebase services through REST APIs.
  - **WebSockets**: For real-time updates between the app and Firebase.
  - **gRPC or Web APIs**: Used within Docker containers for interacting with Firebase for data processing tasks.
  - **Firebase SDKs**: Integrated into both the Flutter app and the Docker backend for simplified communication and data operations.

4. Communication Standards

- **HTTP/HTTPS**: Standard protocol for secure communication between the Flutter app and Firebase.
- **Firebase Realtime Database/WebSocket Protocol**: For real-time data synchronization between the client (Flutter app) and Firebase.
- **Docker Networking**: Docker containers may communicate with external networks using standard HTTP/HTTPS to interact with Firebase.

5. Security and Encryption

- **Authentication**:
  - **Firebase Authentication**: Used to manage user sign-in and sign-out securely.
- **Data Encryption**:
  - **HTTPS**: Ensures all communications between the Flutter app and Firebase are encrypted using SSL/TLS.
  - **Firebase Firestore Rules**: Configured to enforce data access permissions and ensure user data protection.
- **Docker Container Security**:
  - Containers use environment variables to securely store API keys and credentials.
- **Database Security**:
  - **Firestore Security Rules**: Set to protect data, allowing read/write access based on authenticated user roles.
- **Data Encryption at Rest**: Firebase ensures data is encrypted at rest and in transit.

6. Data Transfer Rates and Synchronization Mechanisms

- **Data Transfer Rates**:
  - The data transfer rate depends on network connectivity but is optimized for low latency due to Firebase's real-time capabilities.
  - Typical rates range from a few kilobytes per second for user interactions to larger bursts during data synchronization or bulk updates.
- **Synchronization Mechanisms**:
  - **Real-time Database Synchronization**: Firebase automatically syncs data between the client and server in real time, ensuring up-to-date car park and user data.
  - **Scheduled Data Updates**: The Docker container runs cron jobs or timed tasks to fetch and update car park data in Firebase, ensuring data accuracy.
  - **Push Notifications**: Firebase Cloud Messaging can be utilized for pushing updates or alerts to users regarding car park statuses.

7. Electronic Forms and Communication

- **User Preferences Form**: Stored and updated in Firebase through structured JSON data. User inputs are transmitted via HTTP requests from the Flutter app.
- **Bug Report Form**: Submitted through HTTP requests from the Flutter app to Firebase for logging and processing.

8. Synchronization Mechanisms

- **Backend Processing**:

    - Dockerized Python scripts may use RESTful API calls to Firebase or integrate the Firebase Admin SDK for seamless data processing and updates.
- **Data Consistency**:
    - Firebase's real-time database handles data consistency and synchronization between clients automatically, ensuring that all users see up-to-date information.

9. Communication Security Concerns

- **API Key Protection**: API keys used for Firebase services are stored securely in environment variables within Docker containers.
- **Access Control**: Firebase Firestore security rules are configured to ensure data access is restricted to authenticated users.

10. Secure Communication:

- **SSL/TLS**: All data exchanged between the app, Docker containers, and Firebase uses encrypted channels to prevent interception.
- **Authentication Tokens**: Managed through Firebase to handle user sessions securely.

4. System Features

The functional requirements are categorized by system features, detailing the major services provided by the product.

4.1 Car Park Availability Display

4.1.1 Description and Priority

The Car Park Availability Display feature allows users to view real-time car park availability on a map or list view. This feature is essential for the core functionality of TransitEase, helping users quickly find available parking spots near their current location.

***Priority**: High.*

4.1.2 Stimulus/Response Sequences

1. **User Action**: The user opens the app and grants location permissions.
    - **System Response**: The app retrieves the user's current location and displays it on the map.
2. **User Action**: The user selects the option to view car parks.
    - **System Response**: The app fetches real-time car park data from Firebase (populated by the backend) and displays nearby car parks on the map with availability indicators.

3. **User Action:** The user clicks on a car park icon or name.
    - **System Response**: The app displays detailed information about the selected car park, including total parking spots and available spots.

4.1.3 Functional Requirements

REQ-1: The system must retrieve real-time car park data from Firebase on an interval request basis.
REQ-2: The system must display car parks on a map view based on the user's current location.
REQ-3: The system shall use colored markers to indicate availability.
REQ-4: The system should allow users to switch between map and list views for car park availability.
REQ-5: The system should refresh car park data whenever the user manually refreshes.
REQ-6: If the data retrieval fails (e.g., no network), the system must display an error.

4.2 User Preferences Management

4.2.1 Description and Priority

The User Preferences Management feature allows users to customize their experience by setting preferred car parks, notification preferences, and other settings. This feature enhances user experience by tailoring the app to individual needs.

***Priority**: Medium.*

4.2.2 Stimulus/Response Sequences

1. **User Action**: The user navigates to the Preferences screen.
    - **System Response**: The app loads the user's saved preferences from Firebase and displays them on the screen.
2. **User Action**: The user modifies a preference, such as setting a preferred car park or enabling notifications.
    - **System Response**: The app saves the updated preferences to Firebase.

4.2.3 Functional Requirements

REQ-7: The system should allow users to view and modify their preferences through the Preferences screen.

REQ-8: The system should save changes to preferences in Firebase whenever the user updates them.

REQ-9: The system should allow users to set preferred car parks, which are highlighted when displayed on the map.

REQ-10: The system must enable or disable notifications based on user preferences.

REQ-11: If saving preferences fails, the system must display an error message and allow the user to retry.

4.3 User Authentication

4.3.1 Description and Priority

The User Authentication feature enables secure access to the app's functionalities through Firebase Authentication, allowing users to register, log in, and log out. This feature is necessary for personalizing the app and securing user-specific data.

***Priority**: High.*

4.3.2 Stimulus/Response Sequences

1. **User Action**: The user opens the app and is prompted to log in or sign up.
    - **System Response**: The app displays the login or registration form.
2. **User Action**: The user submits login credentials.
    - **System Response**: The app authenticates the user via Firebase Authentication and grants access if credentials are correct.
3. **User Action**: The user logs out.
    - **System Response**: The app clears session data and returns the user to the login screen.

4.3.3 Functional Requirements

REQ-12: The system must authenticate users through Firebase Authentication for login, registration, and logout functions.

REQ-13: The system must display a login form with fields for email and password, and an option to create an account.

REQ-14: The system must display error messages for invalid login attempts (e.g., incorrect password, non-existing user).

REQ-15: The system must allow users to reset their password via Firebase if forgotten.

REQ-16: Upon logout, the system must clear user session data to prevent unauthorized access.

4.4 Bug Reporting

4.4.1 Description and Priority

The Bug Reporting feature allows users to report issues or feedback directly through the app, which is then stored in Firebase for the development team to review. This feature helps improve the app's quality and reliability.

***Priority**: Medium.*

4.4.2 Stimulus/Response Sequences

1. **User Action**: The user opens the Bug Report screen.
    - **System Response**: The app displays a form with fields for a description and priority level.
2. **User Action**: The user submits the bug report.
    - **System Response**: The app saves the bug report in Firebase and confirms submission to the user.

4.4.3 Functional Requirements

REQ-17: The system should provide a Bug Report screen where users can submit feedback or report issues.

REQ-18: The system should allow users to enter a description of the issue and select a priority level.

REQ-19: The system must save bug reports to Firebase for admin access.

REQ-20: Upon successful submission, the system must display a confirmation message to the user.

REQ-21: If the submission fails, the system must display an error message and allow the user to retry.

4.5 Data Refresh and Synchronization

4.5.1 Description and Priority

The Data Refresh and Synchronization feature ensures that car park data and user preferences are up-to-date, providing a consistent and real-time experience across devices.

***Priority**: High.*

4.5.2 Stimulus/Response Sequences

1. **System Action**: The backend periodically fetches updated car park data from the URA Carpark API.
    - **System Response**: The backend processes and pushes the updated data to Firebase.
2. **User Action**: The user manually refreshes car park data in the app.
    - System Response: The app retrieves the latest data from Firebase and updates the display.

4.5.3 Functional Requirements

REQ-22: The backend must retrieve updated car park data from the URA Carpark API every set interval.

REQ-23: The backend must process the data and push it to Firebase.

REQ-24: The app must automatically fetch the latest data from Firebase when the user opens it.

REQ-25: The system must allow the user to manually refresh data by pulling down on the main dashboard.

REQ-26: If data retrieval fails, the system must display an error message and retry after 1 minute.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Response Time:

- The TransitEase app must retrieve and display car park availability data within **2 seconds** on a stable network connection (e.g., 4G or Wi-Fi).
- User actions, such as opening the Preferences screen or submitting a bug report, should respond within **2 second**.

Data Refresh Rate:

- Car park availability data should be refreshed every **10 minutes** to provide users with up-to-date information.

Large User Base:

- The system should support up to **10,000 concurrent users** without performance degradation. Firebase Firestore must handle simultaneous data requests for car park data and user preferences without delays.

Load Time:

- The app's initial load time should not exceed **3 seconds** on modern devices. Backend responses from Firebase and the URA API should be optimized to support this requirement.

Reliability and Reliability:

- The system architecture should allow for redundancy. Should the API fail, the flutter application should **still be able to access** carpark data stored on Firestore Database and **login/sign-up** using Firebase Authentication.

- The application **should not stop** giving users the location and last available rates for the car park.

5.2 Safety Requirements

Data Integrity:

- The app must ensure the integrity of car park data fetched from the URA API and preferences stored in Firebase to prevent users from making parking decisions based on inaccurate or outdated information.

User Action Safety:

- The app must prevent users from performing sensitive actions (e.g., updating preferences or submitting reports) if the app detects potential data corruption or connection issues. In such cases, the system must display an error message to prevent incomplete transactions.

Compliance:

- TransitEase must comply with all relevant safety and regulatory guidelines in Singapore to ensure safe use of real-time data, particularly around car park availability.

5.3 Security Requirements

User Authentication:

- The system must require secure user authentication through Firebase Authentication. Users must log in with email and password, and passwords must adhere to minimum complexity requirements.
- Password reset functionality should include verification through email to prevent unauthorized access.

Data Privacy and Protection:

- All user data, including preferences and bug reports, must be stored in compliance with **GDPR** and **PDPA** (Personal Data Protection Act in Singapore).
- Data in transit (between app, backend, Firebase, and URA API) must be encrypted using **TLS/SSL**.

Access Control:

- Only authenticated users must have access to the app's functionalities, such as viewing car park data, updating preferences, and submitting bug reports.
- Access to Firebase must be controlled with Firebase Security Rules to restrict unauthorized access and modification.

Audit Logs:

- The system must maintain logs of sensitive operations, such as login attempts, failed authentication, and preference updates, to facilitate security audits and incident response.

5.4 Software Quality Attributes

Reliability:

- The app should operate with at least **99.5% uptime** for data services (Firebase and URA API), ensuring consistent data access for users.

Usability:

- The interface should be intuitive and user-friendly, with easy navigation and clear labels. Users should be able to perform basic actions (e.g., check car park availability, log in, set preferences) without a learning curve.
- The app should include concise error messages and help texts to guide users through troubleshooting.

Maintainability:

- The backend code (Python) and app code (Flutter) must be structured and documented to allow easy updates and maintenance, especially for bug fixes and API changes.
- The app should be easily modifiable for future feature additions, such as adding support for new regions or integrating with additional APIs.

Portability:

- The TransitEase app should be compatible with iOS (version 13+)

Interoperability:

- The app must seamlessly interact with the URA Carpark API and Firebase services, ensuring smooth data flow between components without conflicts or data mismatches.

Testability:

- Each feature should be designed to allow easy testing, with test cases covering both functional (e.g., car park availability) and non-functional (e.g., load time, response time) aspects.

5.5 Business Rules

User Access and Privileges:

- Only registered users can access core functionalities such as checking car park availability, saving preferences, and submitting bug reports.
- Non-registered users may only access the splash screen and login/registration functionalities.

Data Update and Refresh:

- Car park availability should be updated based on the latest data from the URA Carpark API. The backend must manage data requests to avoid API rate limits, caching data when necessary for faster access.

Feedback and Bug Reporting:

- Users must be logged in to submit bug reports or feedback. Bug reports should include a brief description and priority level to help developers prioritize issues.

Data Retention Policy:

- User preferences and car park data should be retained in Firebase for a minimum of **6 months**. Users should have the option to delete their preferences upon account deactivation.

6. Other Requirements

6.1 Database Requirements

- **Data Storage**:

    - The app must use **Firebase Firestore** as the primary database to store user preferences, car park availability data, bug reports, and user authentication information.
- **Data Consistency**:
    - Car park availability data must be updated in Firebase every **5 minutes** to reflect real-time availability from the URA Carpark API. This ensures that users receive accurate and up-to-date information.
    - The app must handle data synchronization effectively between Firebase and local device storage, especially in cases of intermittent internet connectivity.

6.2 Globalization Requirements

- **Language Support**:
    - The initial release of TransitEase will support **English** as the default language.
    - Future versions may consider supporting additional languages based on user demand and regional adoption (e.g., **Mandarin** for broader Singaporean use).
- **Date and Time Formatting**:
    - All date and time information displayed in the app (e.g., last updated timestamps for car park data) must follow the **24-hour format** and **DD-MM-YYYY** date format, as these are standard in Singapore.
    - The app must consider user device settings to automatically format dates and times according to local conventions when internationalization is expanded.

6.3 Legal Requirements

- **Data Privacy**:
    - The app must comply with the **Personal Data Protection Act (PDPA)** in Singapore, ensuring that all personal information collected (e.g., user preferences and authentication details) is securely stored and only used for its intended purpose.
- **User Consent**:
    - Users must consent to the app's privacy policy before registration, clearly explaining the types of data collected, how it will be used, and user rights..
- **API Usage Compliance**:
    - The app must comply with the usage terms of the **URA Carpark API**, including adhering to rate limits and ensuring data is presented accurately to end-users without manipulation.

6.4 Reuse Objectives

- **Code Reusability**:
    - The TransitEase backend code (written in Python) must be modular and reusable to support potential expansion to other cities or regions if additional car park data APIs become available.
    - Uses **Flutter** for cross-platform compatibility (iOS and Android) to maximize code reuse and minimize platform-specific development efforts.
- **Reusable Components**:

- The authentication and user preferences management modules, implemented via Firebase, must be designed for easy reuse across other applications with minimal modification.

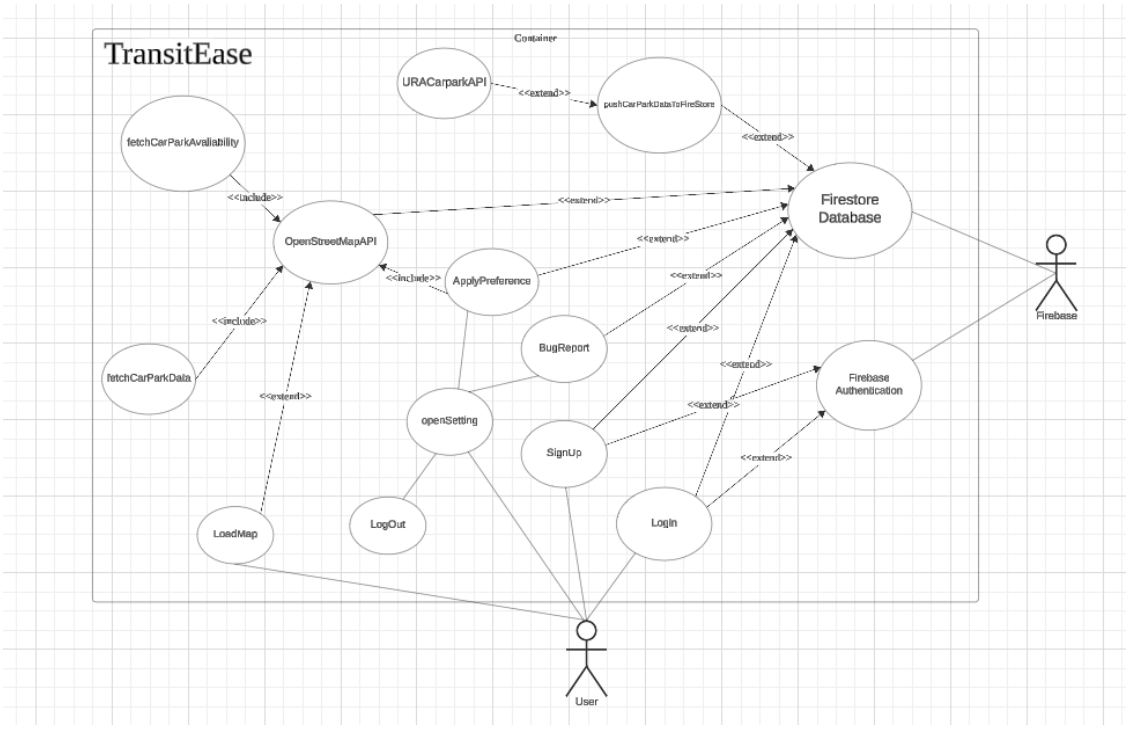
Appendix A: Glossary

API (Application Programming Interface)	A set of functions and protocols that allow software applications to communicate with each other.
Firebase	A platform developed by Google for creating mobile and web applications, used in TransitEase for authentication and database management.
Firestore	A NoSQL database by Firebase, used to store and manage user preferences, car park data, and other app-related information.
Flutter	An open-source UI software development kit created by Google, used for developing TransitEase as a cross-platform mobile application.
GPS (Global Positioning System)	A satellite-based navigation system used to determine the user's location.
PDPA (Personal Data Protection Act)	A Singaporean law governing the collection, use, disclosure, and care of personal data.
URA (Urban Redevelopment Authority)	A statutory board under the Ministry of National Development of the Singapore Government, which provides car park availability data via the URA Carpark API.

Appendix B: Analysis Models

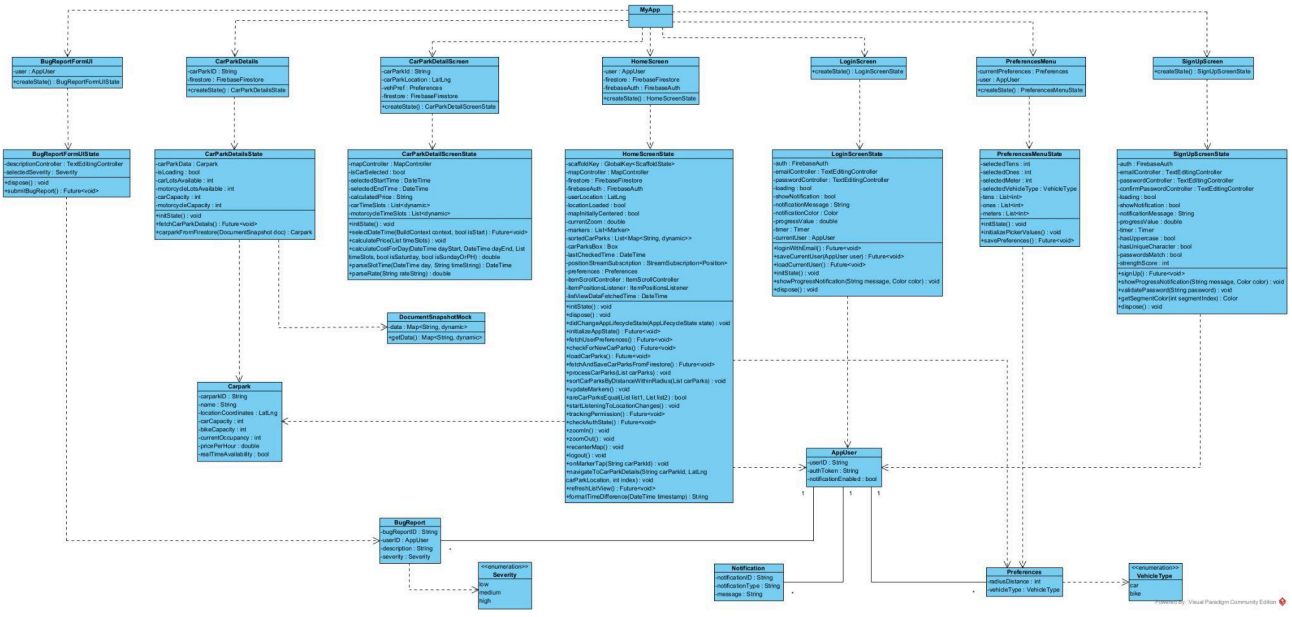
The Analysis Models below can also be found in the Lab5Finals PDF document in the github repository.
This section includes diagrams relevant to the analysis and understanding of the TransitEase system:

1. **UseCase Diagram**
   - Depicts the flow of data within the TransitEase app, including user inputs, data retrieval from the URA Carpark API, and storage in Firebase.

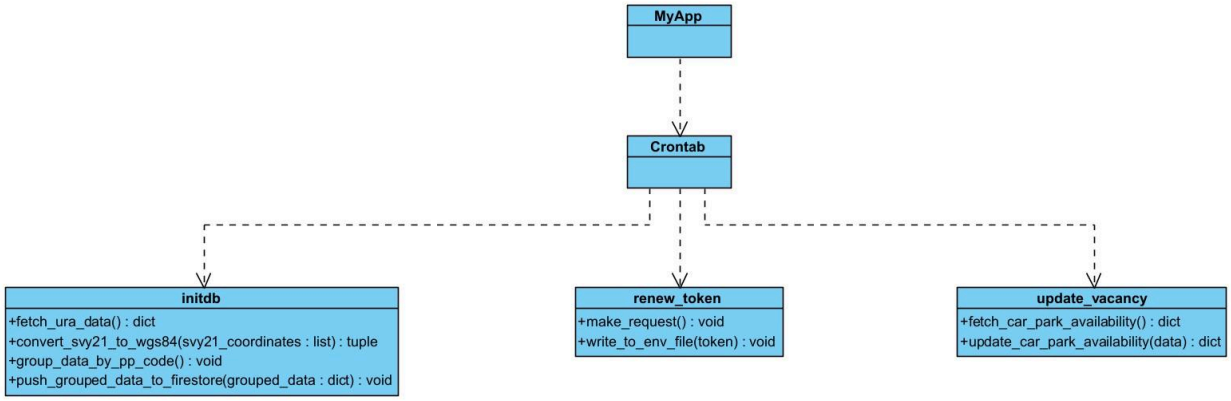


2. **Class Diagram**:
   - Represents the main classes in the TransitEase app, such as User, CarPark, and Preferences, and their relationships.

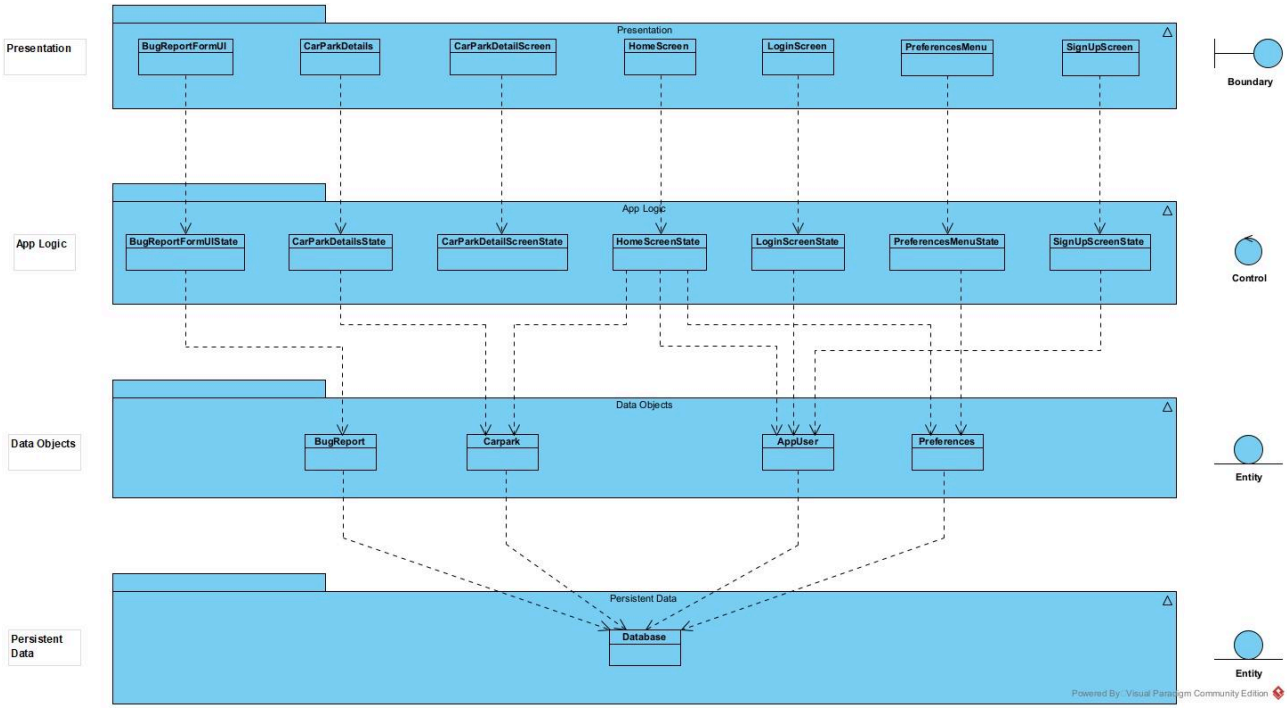
   2.1. **Frontend Class Diagram:**



2.2. **Backend Class Diagram:**

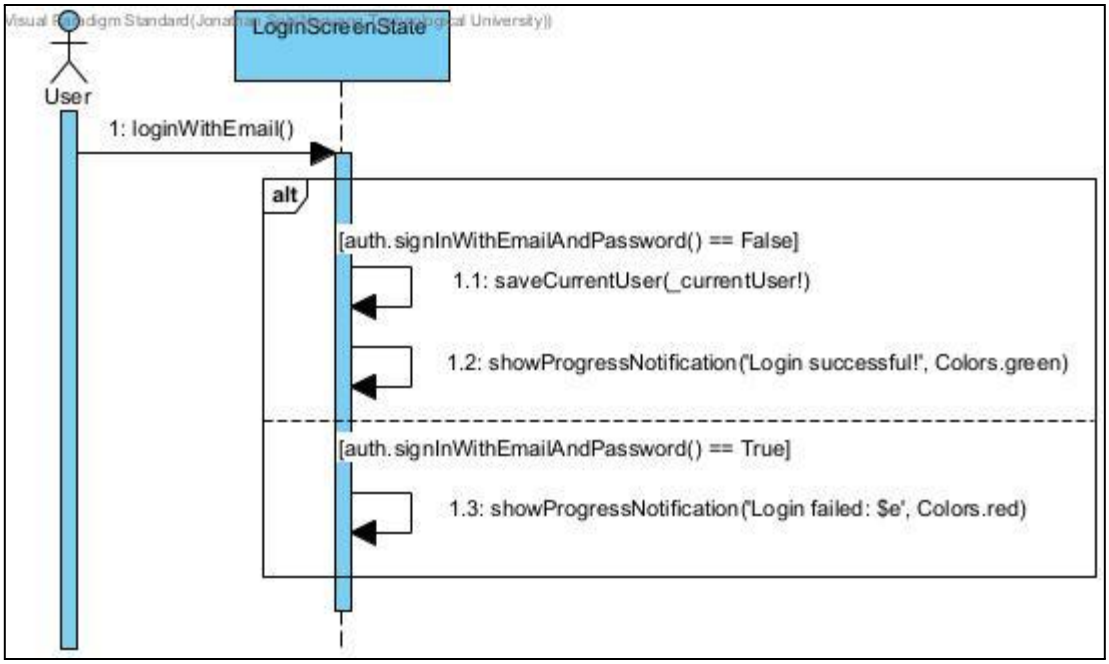


3. **System Architecture Diagram:**
   - Overview of the primary components and interactions within the system

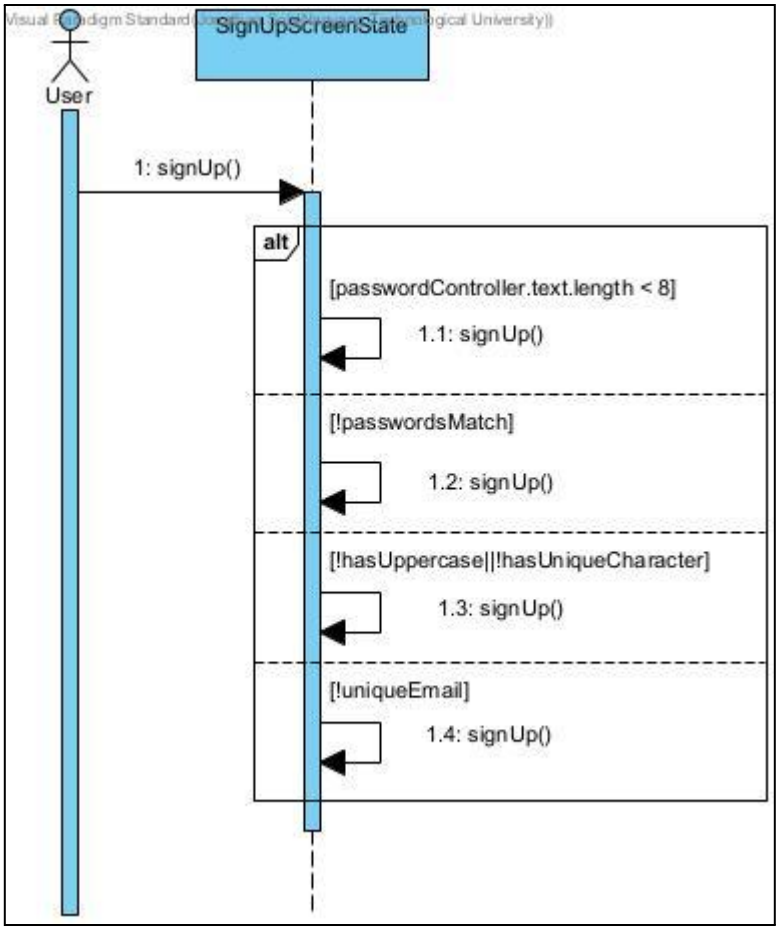


4. **Sequence Diagram:**
   - Shows the step-by-step interactions between different system components during specific use cases, illustrating the order of operations and data flow in detail.

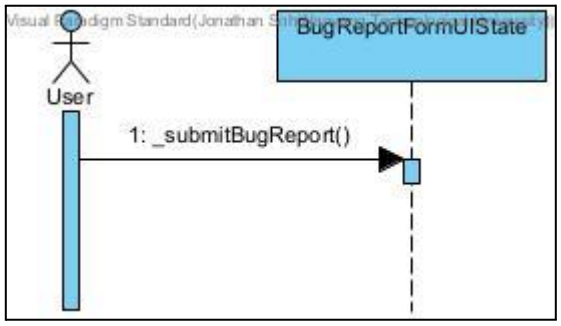
4.1. **Login:**


LoginScreenState
User
1: loginWithEmail()
alt
[auth.signInWithEmailAndPassword() == False]
1.1: saveCurrentUser(_currentUser!)
1.2: showProgressNotification('Login successful!', Colors.green)
[auth.signInWithEmailAndPassword() == True]
1.3: showProgressNotification('Login failed: $e', Colors.red)

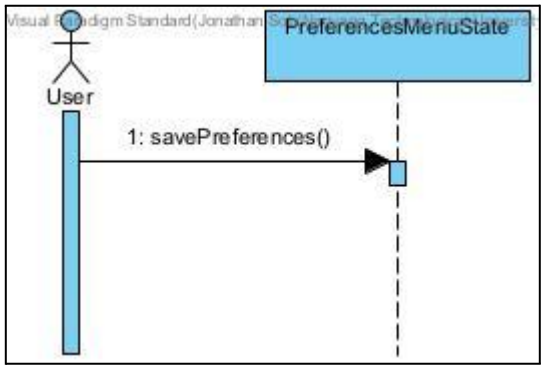
4.2. **SignUp:**


SignUpScreenState
User
1: signUp()
alt
[passwordController.text.length < 8]
1.1: signUp()
[!passwordsMatch]
1.2: signUp()
[!hasUppercase||!hasUniqueCharacter]
1.3: signUp()
[!uniqueEmail]
1.4: signUp()

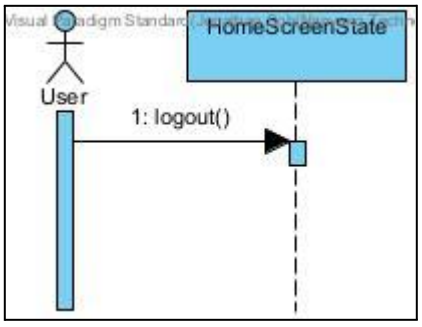
4.3. **ReportBug:**



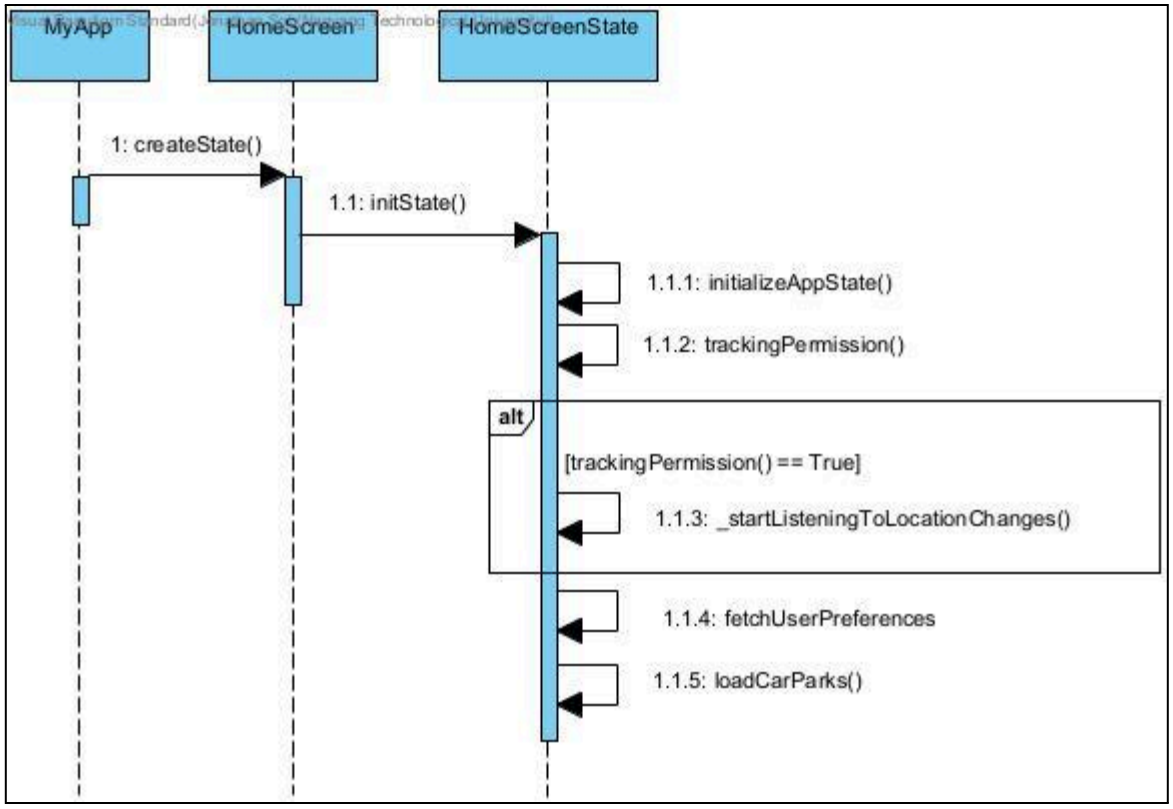
4.4. **ApplyPreference:**



4.5. **LogOut:**



4.6. **LoadMap:**

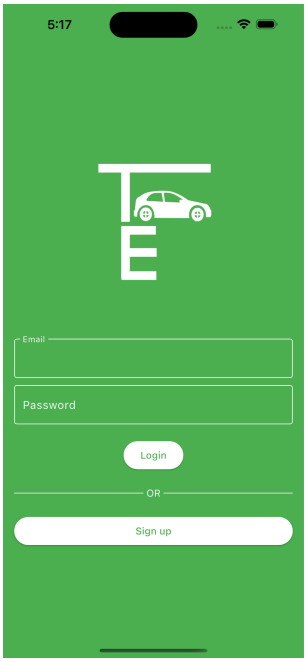


Appendix C: To Be Determined List

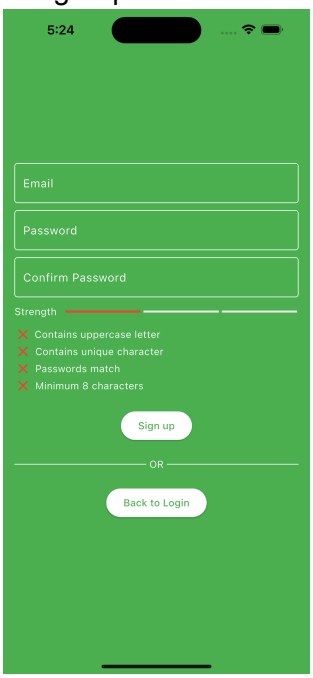
1. **Additional Language Support**: Decide on additional languages to support beyond English, based on initial user feedback and demand

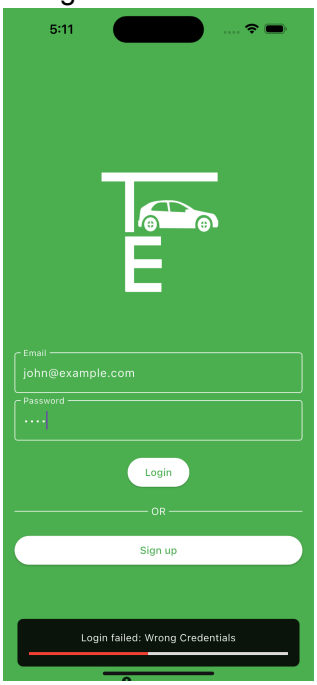
Appendix D: Figures

- Figure 1 : Login Screen



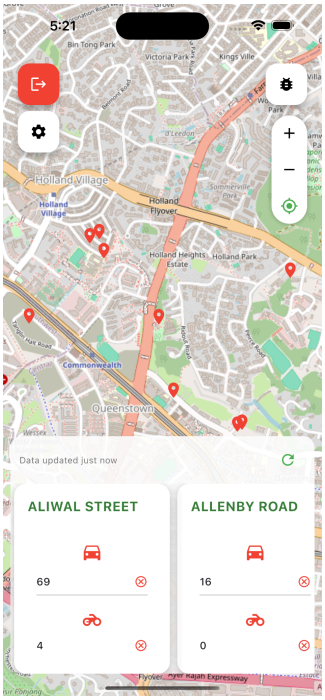
- Figure 2: Sign-up Screen



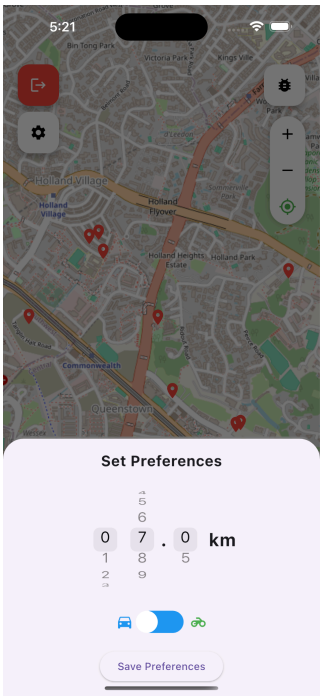
- Figure 3: Login Fail
  - 

- Figure 4: Login Pass
  - 

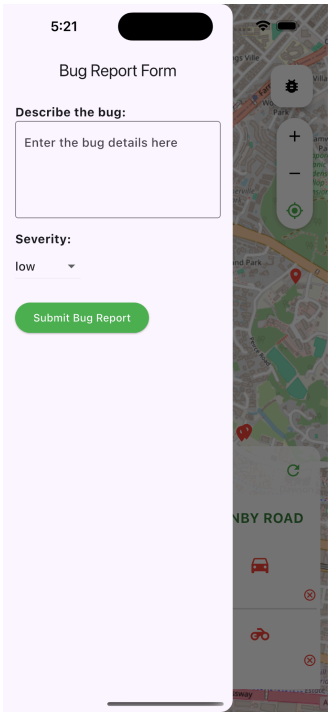
- Figure 5: Home Screen



- Figure 6: Preferences Menu



- Figure 7: Bug Report Form



- Figure 8: Carpark Details Screen