~~Use Cases~~ Lab 2

for

TransitEase

Version 2.0 approved

Prepared by Ashwin, Dave, Jun Heng, Jonathan

Nanyang Technological University

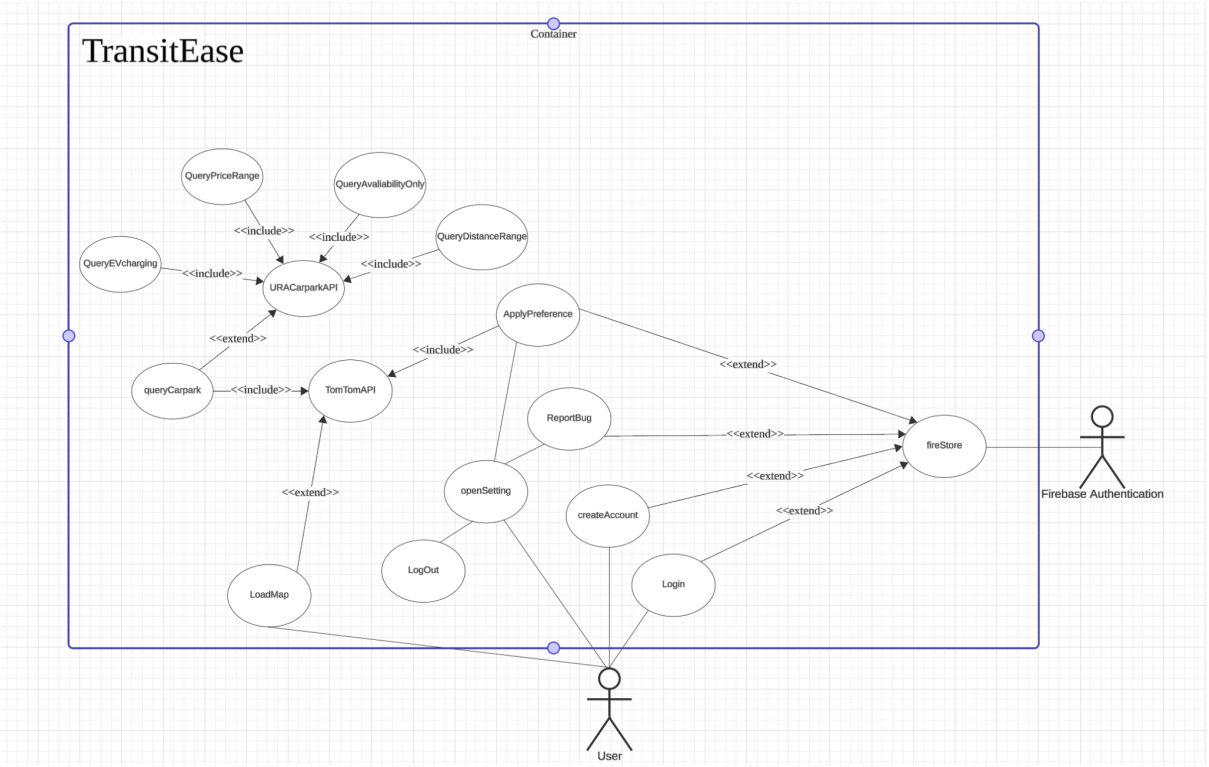
16/09/24

Revision History

Name	Date	Reason For Changes	Version
Ashwin	17 Sept	Added Boundary Classes	2.0
Dave	17 Sept	Added Initial Dialog Map	2.0
Jonathan	17 Sept	Added class diagram of entity classes	2.0
Goh Jun Heng	17 Sept	Added Class Diagram of entity classes	2.0

1. Complete Use Case diagram....3
2. Use Case Descriptions....4
3. Class diagram of entity classes....12
4. Key boundary classes and control classes....12
4.1. Boundary Classes....12
Authentication UI....12
CarparkListViewUI....13
PreferenceFormUI....13
SwapUI....14
MapViewUI....14
BugReportFormUI....14
4.2. Control Classes....15
AuthenticationController....15
UserPreferenceController....16
CarparkController....16
LocationController....17
FirebaseController....17
BugReportController....18
5. Sequence diagrams of some use cases....18
6. Initial Dialog map....24

1. Complete Use Case diagram



2. Use Case Descriptions

Use Case ID:	1		
Use Case Name:	LogInEmail		
Created By:	Dave Goh	Last Updated By:	Dave Goh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Allows users to log in using an existing account via email credentials.
Preconditions:	The user must have already registered an account.
Postconditions:	The user gains access to the app's main interface upon successful login.
Priority:	High
Frequency of Use:	1
Flow of Events:	1. User selects "Login with Email" in the user interface. 2. User enters an email and password. 3. System validates credentials with Firebase authentication. 4. Users are granted access upon successful authentication.
Alternative Flows:	NIL
Exceptions:	1. Incorrect password. 2. Username does not exist.
Includes:	Firebase authentication API
Special Requirements:	Integration with Firebase Authentication.
Assumptions:	User has already created an account before.
Notes and Issues:	None

Use Case ID:	2		
Use Case Name:	LoginWGoogle		
Created By:	Ashwin Suresh	Last Updated By:	Ashwin Suresh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Enables users to log in using their Google account credentials.
Preconditions:	The user must have a valid Google account.
Postconditions:	The user gains access to the app's main interface upon successful login.
Priority:	High
Frequency of Use:	Once per session
Flow of Events:	1. User selects "Login with Google" in the user interface. 2. User is prompted to grant app permissions to access their Google account. 3. System authenticates user via Google authentication. 4. User logs in successfully.
Alternative Flows:	NIL
Exceptions:	User denies app permission to access their Google account.
Includes:	
Special Requirements:	Integration with Google Authentication.
Assumptions:	User has a valid and accessible Google account.
Notes and Issues:	None

Use Case ID:	3		
Use Case Name:	CreateAccount		
Created By:	Ashwin Suresh	Last Updated By:	Ashwin Suresh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Allows new users to create an account using their email.
Preconditions:	The email must not be registered in the system.

Postconditions:	A new account is created, and the user can log in with their credentials.
Priority:	High
Frequency of Use:	Typically used once by new users
Flow of Events:	User selects "Create Account" in the user interface. User enters a valid email, password, and confirms the password. System checks for existing account using Firebase API. Account is created successfully if email is unique.
Alternative Flows:	NIL
Exceptions:	1. Email already exists. 2. Passwords do not match.
Includes:	Firebase API
Special Requirements:	Integration with Firebase.
Assumptions:	1. User provides a valid email. 2. User inputs matching passwords.
Notes and Issues:	None

---

4. Load Map

Use Case ID:	4		
Use Case Name:	LoadMap		
Created By:	Ashwin Suresh	Last Updated By:	Ashwin Suresh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Loads the map interface showing the user’s current location.
Preconditions:	User must have granted location permissions.

Postconditions:	The map displays the user’s location.
Priority:	High
Frequency of Use:	Every time the map is loaded
Flow of Events:	1. User is prompted to grant location permission if not already granted. 2. User grants permission. 3. Map is loaded and centred on the user's current location. 4. User's location is indicated on the map.
Alternative Flows:	NIL
Exceptions:	User denies location permission. No network connectivity.
Includes:	TomTomAPI
Special Requirements:	Location permission must be granted
Assumptions:	The user has internet access
Notes and Issues:	None

---

5. Query Nearby Car Parks

Use Case ID:	5		
Use Case Name:	queryNearbyCarpark		
Created By:	Dave Goh	Last Updated By:	Dave Goh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	System
Description:	Queries and displays nearby carparks based on the user’s location.
Preconditions:	Network connectivity and location permissions are granted.
Postconditions:	The user can view nearby carparks with details.
Priority:	High

Frequency of Use:	Frequently used when searching for parking.
Flow of Events:	1. System queries URA Carpark API for nearby carpark information based on the user's location. 2. Car Parks are displayed on the map in proximity to the user's location. 3. Car Park information such as distance, rate, EV charging capability, and capacity are displayed.
Alternative Flows:	NIL
Exceptions:	1. No network connectivity. 2. Location permissions are not granted.
Includes:	1. URA Carpark API 2. TomTom API
Special Requirements:	Integration with URA and TomTom APIs.
Assumptions:	User has internet access.
Notes and Issues:	None

Use Case ID:	6		
Use Case Name:	reportBug		
Created By:	Ashwin Suresh	Last Updated By:	Ashwin Suresh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Allows users to submit a bug report.
Preconditions:	User must be logged in.
Postconditions:	Bug report is saved in the database for review.
Priority:	High

Frequency of Use:	Used occasionally.
Flow of Events:	1. User navigates to the settings menu. 2. User selects the "Report Bug" option. 3. User enters the bug description in a text field. 4. User submits the bug report. 5. Report is stored in the database.
Alternative Flows:	NIL
Exceptions:	1. Text field contains invalid characters. 2. Report exceeds 1000 characters. 3. No network connectivity.
Includes:	Firebase API
Special Requirements:	Integration with Firebase.
Assumptions:	User has internet access.
Notes and Issues:	None

---

7. Change Preferences

Use Case ID:	7		
Use Case Name:	changePreferences		
Created By:	Dave Goh	Last Updated By:	Dave Goh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Allows users to modify their app preferences.
Preconditions:	Users must be on the preferences page.
Postconditions:	Changes are saved to the user’s profile in the database.
Priority:	High

Frequency of Use:	Used frequently to adjust preferences.
Flow of Events:	1. User navigates to the preferences page. 2. Users modify their preferences. 3. System updates the changes in the user database.
Alternative Flows:	NIL
Exceptions:	No network connectivity.
Includes:	None
Special Requirements:	None
Assumptions:	User has internet access
Notes and Issues:	None

---

8. Logout

Use Case ID:	8		
Use Case Name:	Logout		
Created By:	Dave Goh	Last Updated By:	Dave Goh
Date Created:	17/09/24	Date Last Updated:	17/09/24

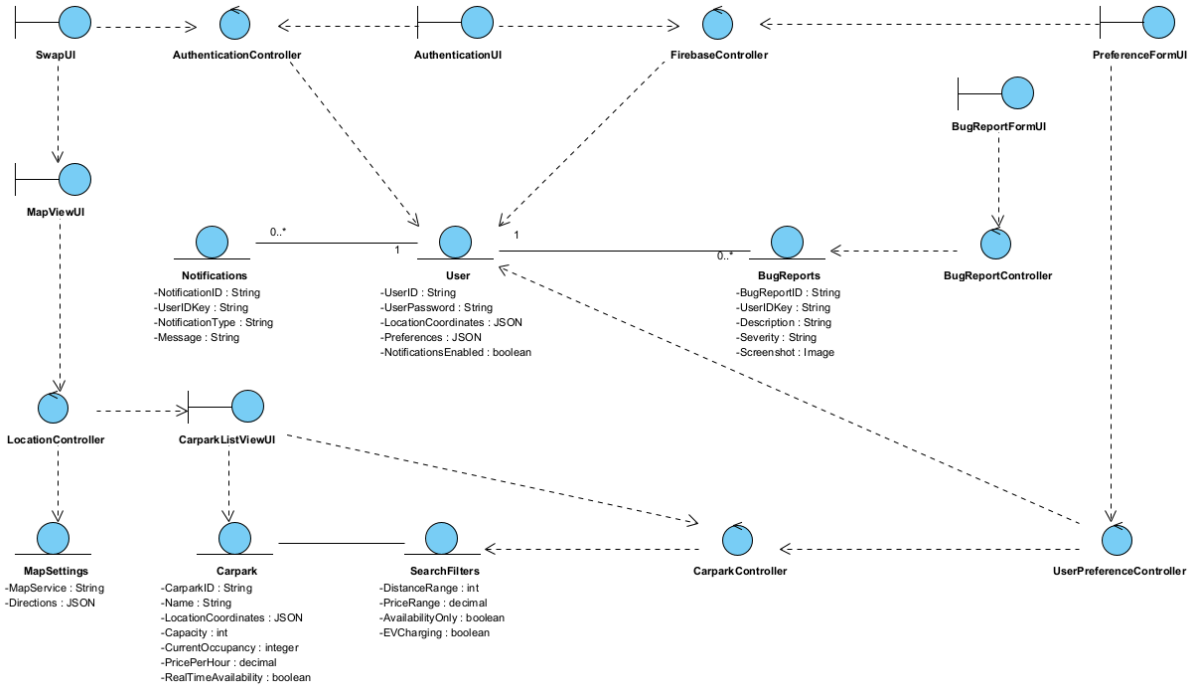
Actor:	User
Description:	Logs the user out of the app.
Preconditions:	The user must be logged in.
Postconditions:	User is redirected to the login screen.
Priority:	High
Frequency of Use:	Frequently used at the end of a session.
Flow of Events:	1. User selects "Logout." 2. Application logs out the user and redirects them to the login page.
Alternative Flows:	NIL
Exceptions:	No network connectivity.
Includes:	None
Special Requirements:	None
Assumptions:	None

Notes and Issues:	None

3. Class diagram of entity classes

3.1. Actual class diagram below

3.2. Conceptual Diagram



4. Key boundary classes and control classes

4.1. Boundary Classes

4.1.1.

Authentication UI
• **Responsibility:** Interacts with the user for login, logout, and sign-up processes using Firebase.
Methods: • `login(email: String, password: String): void` — Presents login form and submits user credentials to Firebase. • `logout(): void` — Handles user logout and updates the UI accordingly.

- `register(email: String, password: String): void` — Presents registration form and registers the user via Firebase.
- `showLoginError(errorMessage: String): void` — Displays login error message to the user.

4.1.2.

CarparkListViewUI
• **Responsibility:** Displays a list of nearby carparks to the user.
Methods: • `displayCarparkList(carparks: List<Carpark>): void` — Renders the list of car parks based on user location. • `displayCarparkDetails(carparkId: String): void` — Opens detailed view of a selected carpark. • `showLoadingIndicator(): void` — Displays a loading animation while fetching carparks. • `showErrorMessage(errorMessage: String): void` — Displays errors when loading carpark data fails.

4.1.3.

PreferenceFormUI
• **Responsibility:** Collects and displays the user's parking preferences.
Methods: • `loadPreferences(preferences: Preferences): void` – Displays existing preferences from Firebase for editing. • `savePreferences(): Preferences` – Gathers the user's preferences from the form and submits them.

- `showSaveSuccess(): void` – Notifies the user when preferences have been successfully saved.
- `showSaveError(errorMessage: String): void` – Notifies the user if saving preferences fails.

4.1.4.

SwapUI

- **Responsibility:** Changes the screen/user interface on the application

Methods:

- `changeLogin(): void` — Brings you to login screen
- `changeMap(): void` — Brings you to main/map screen

4.1.5.

MapViewUI

- **Responsibility:** Displays carparks on a map interface with real-time location updates.

Methods:

- `showCarparkOnMap(carparks: List<Carpark>): void` — Plots the available carparks on the map.
- `showUserLocation(location: LatLng): void` — Displays the current user location on the map.
- `updateCarparkMarkers(carparks: List<Carpark>): void` — Updates carpark markers based on user's new location or preferences

4.1.6.

BugReportFormUI

- **Responsibility:** Provides the interface for users to submit bug reports.

Attributes:

- `descriptionField: String` — Text field where users describe the bug.
- `screenshot: Image` — Allows users to upload an optional screenshot.
- `severityDropdown: String` — Dropdown menu for selecting bug severity (e.g., low, medium, high).

Methods:

- `submitBugReport(): void` — Gathers data from the form and sends it to the controller for processing.
- `displaySubmissionSuccess(): void` — Shows a message indicating successful bug report submission.
- `displaySubmissionError(errorMessage: String): void` — Displays an error message if the submission fails.
- .

4.2. Control Classes

4.2.1.

AuthenticationController

- **Responsibility:** Manages user authentication processes via Firebase

Methods:

- `loginUser(email: String, password: String): boolean` — Authenticates the user via Firebase and returns the status.
- `logoutUser(): void` — Logs out the user and cleans up session data.

- `registerUser(email: String, password: String): boolean` — Registers a new user in Firebase and manages session data.

4.2.2.

UserPreferenceController

- **Responsibility:** Handles the saving and retrieving of user preferences.

Methods:

- `loadPreferences(userId: String): Preferences` — Fetches user preferences from Firebase.
- `savePreferences(userId: String, preferences: Preferences): void` — Saves user preferences to Firebase.
- `getDefaultPreferences(): Preferences` — Returns default preferences for new users.

4.2.3.

CarparkController

- **Responsibility:** Orchestrates carpark fetching and filtering logic.

Methods:

- `getNearbyCarparks(location: LatLng, preferences: Preferences): List<Carpark>` — Fetches nearby carparks based on the user's current location and preferences.
- `filterCarparks(preferences: Preferences, carparks: List<Carpark>): List<Carpark>` — Filters carparks based on the user's preferences (distance, cost, type, etc.).

- `sortCarparksByDistance(carparks: List<Carpark>, location: LatLng): List<Carpark>` — Sorts car parks based on proximity to the user's current location

4.2.4.

LocationController

- **Responsibility:** Manages location updates and provides location data to other controllers.

Methods:

- `getCurrentLocation(): LatLng` — Retrieves the current user location from the device's GPS.
- `startLocationTracking(): void` — Starts continuous tracking of the user's location.
- `stopLocationTracking(): void` — Stops location tracking to conserve resources.

4.2.5.

FirebaseController

- **Responsibility:** Manages data exchange with Firebase for authentication and database operations.

Methods:

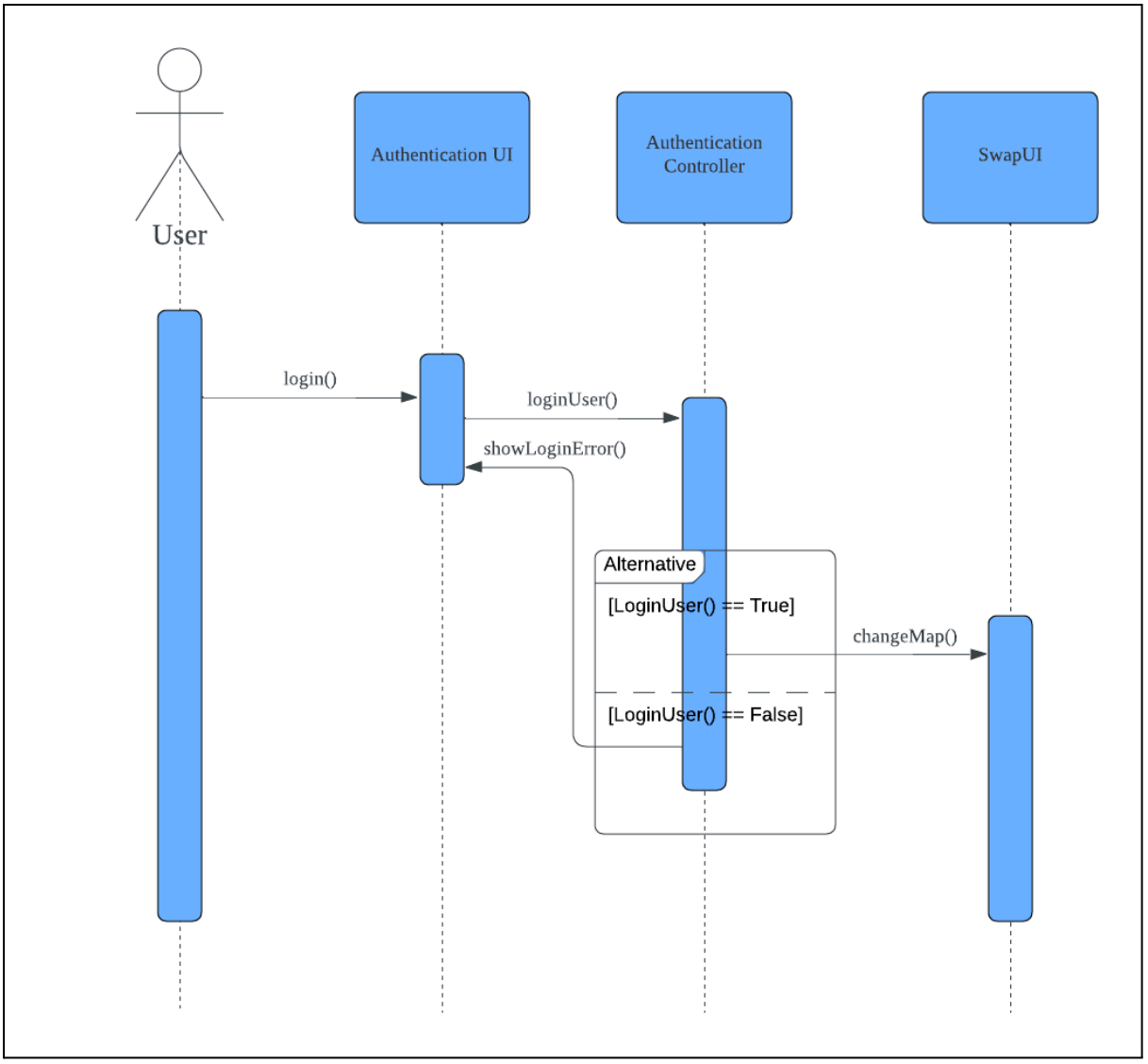
- `saveData(collection: String, documentId: String, data: Map): void` — Saves data to Firebase Firestore.
- `getData(collection: String, documentId: String): Map` — Retrieves data from Firebase Firestore.
- `signIn(email: String, password: String): boolean` — Signs in the user via Firebase Authentication.
- `signOut(): void` — Signs out the user from Firebase.

4.2.6.

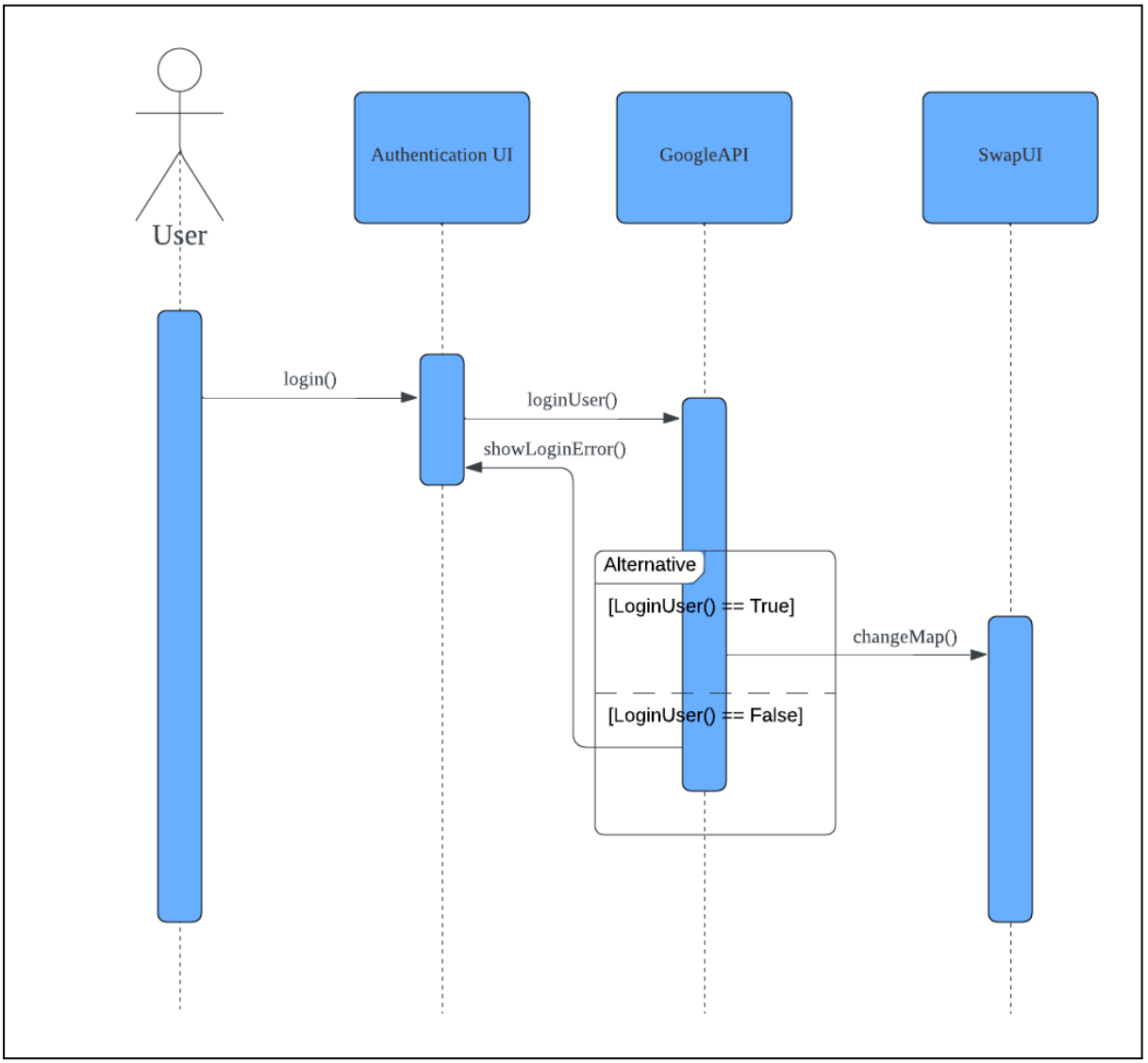
BugReportController
• **Responsibility:** Manages the submission, validation, and storage of bug reports.
`Methods:` • `submitBugReport(description: String, screenshot: Image, severity: String, userId: String): void` – Validates and sends the bug report to Firebase. • `validateBugReport(description: String): boolean` – Ensures that the bug report has a valid description (e.g., not empty).

5. Sequence diagrams of some use cases

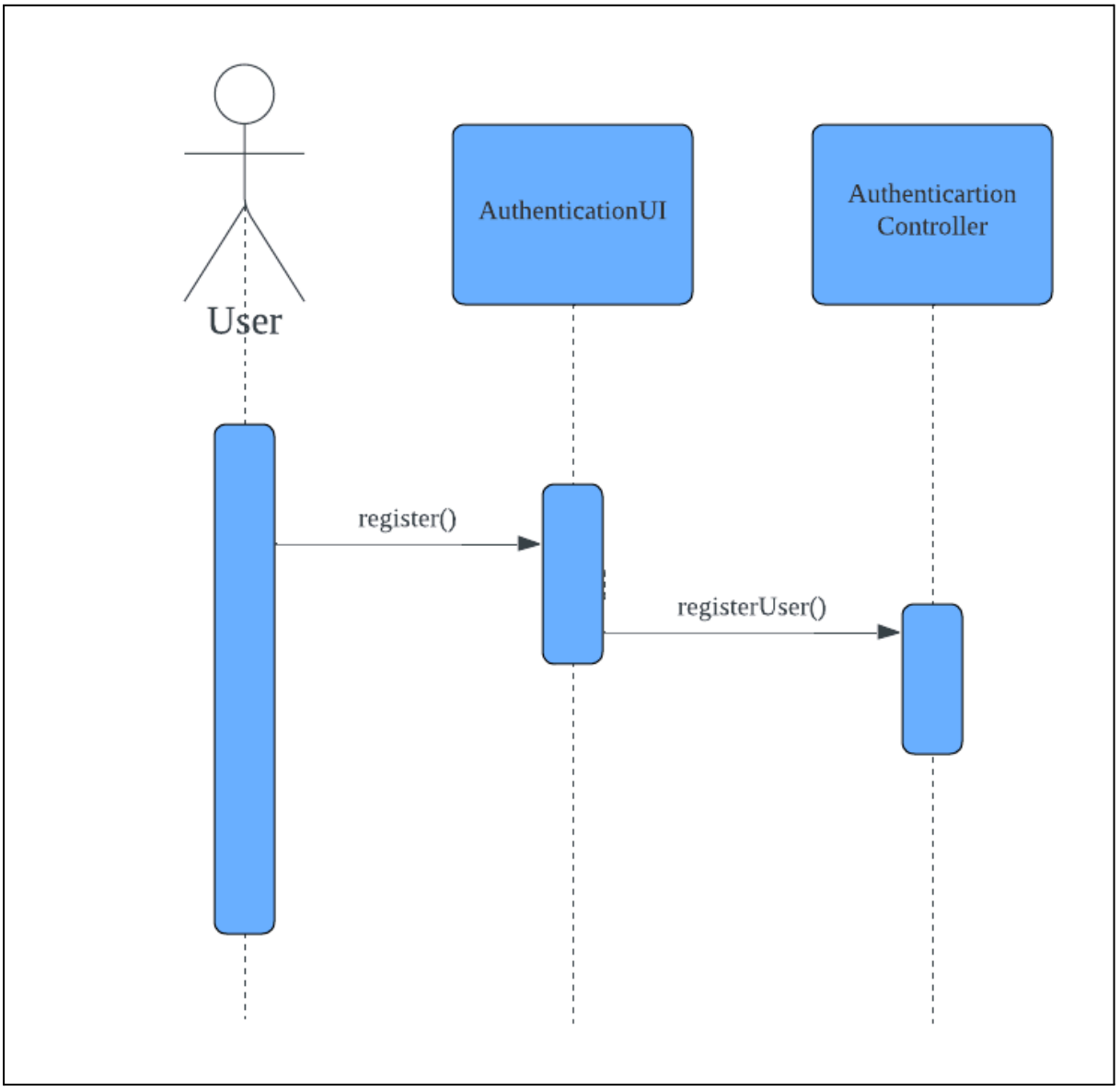
5.1. LogInEmail


User
Authentication UI
Authentication Controller
SwapUI
login()
loginUser()
showLoginError()
Alternative
[LoginUser() == True]
changeMap()
[LoginUser() == False]

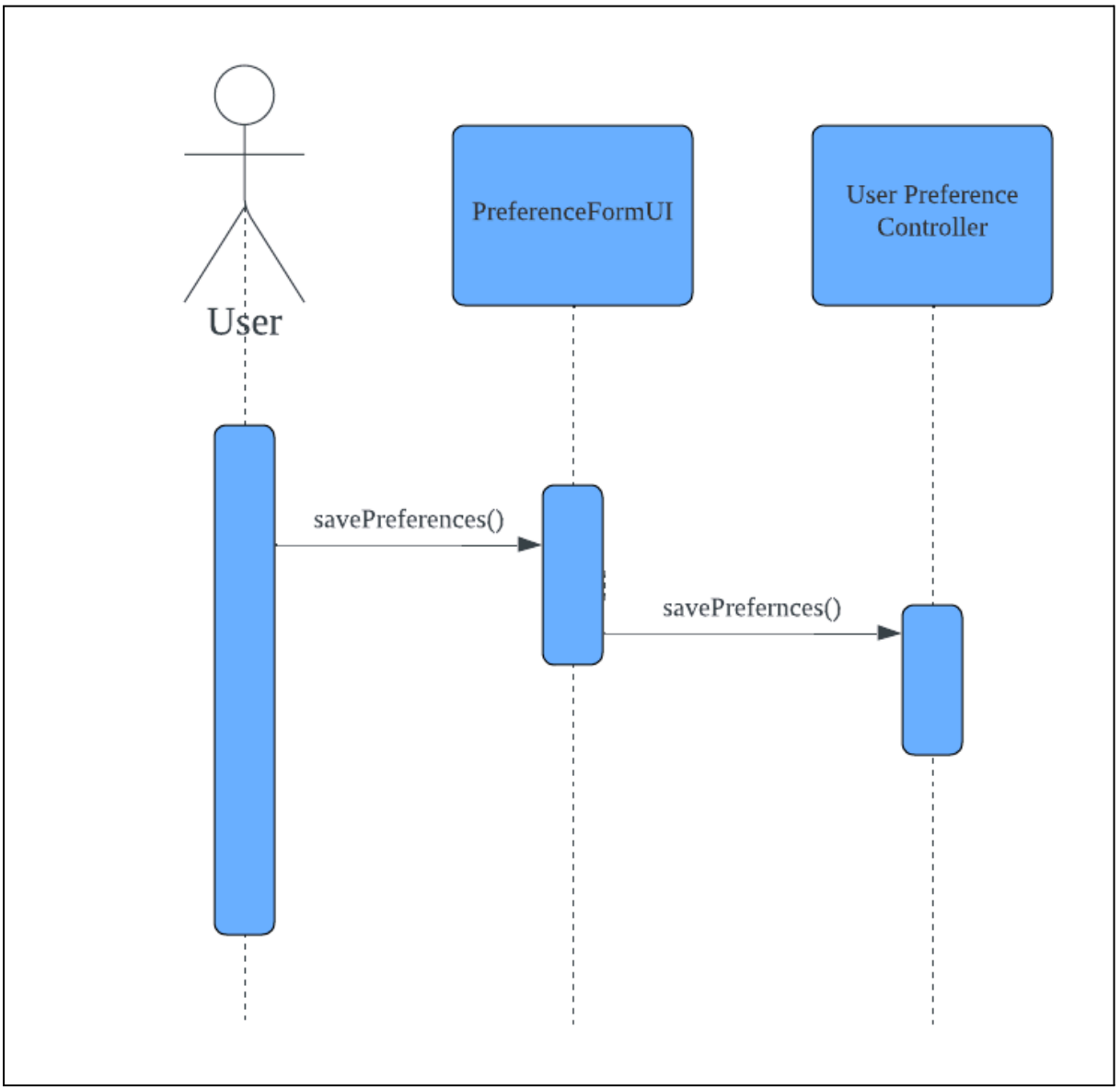
5.2. LoginWGoogle



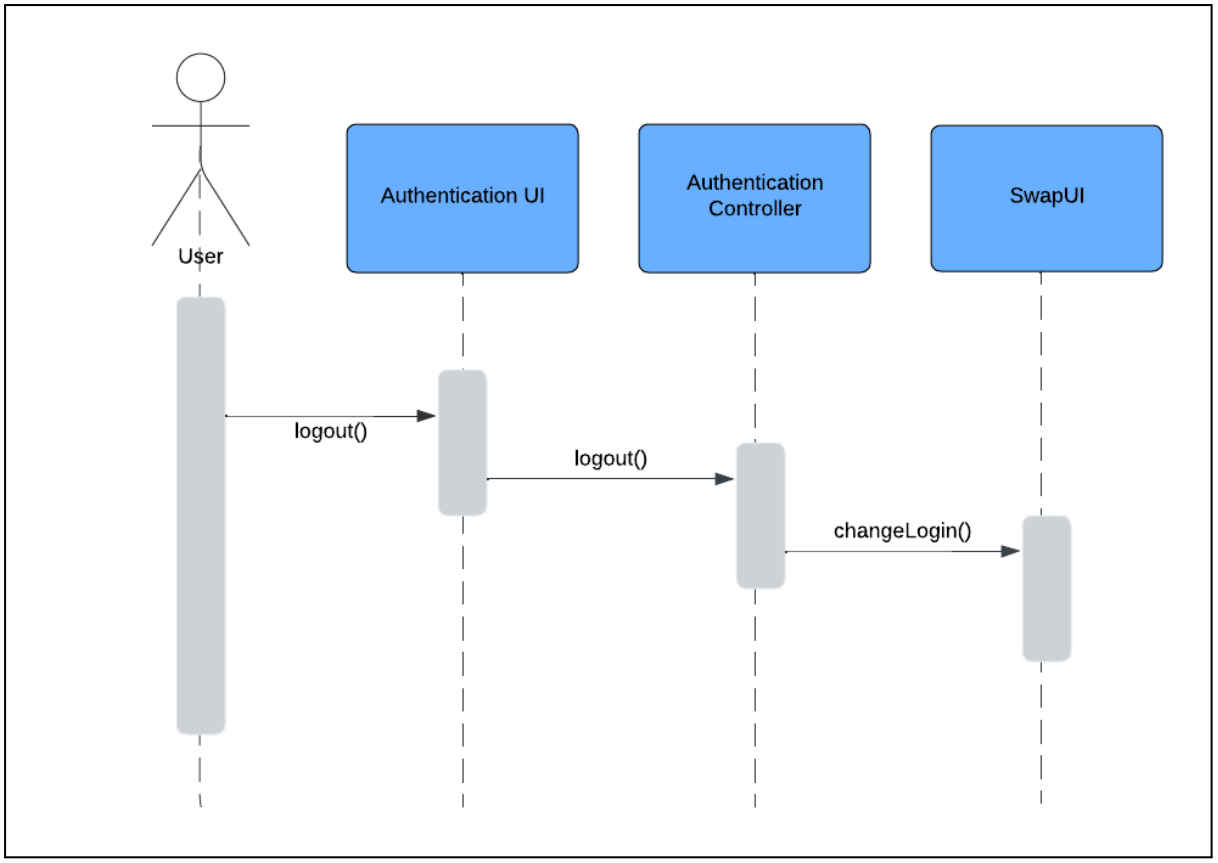
5.3. CreateAccount



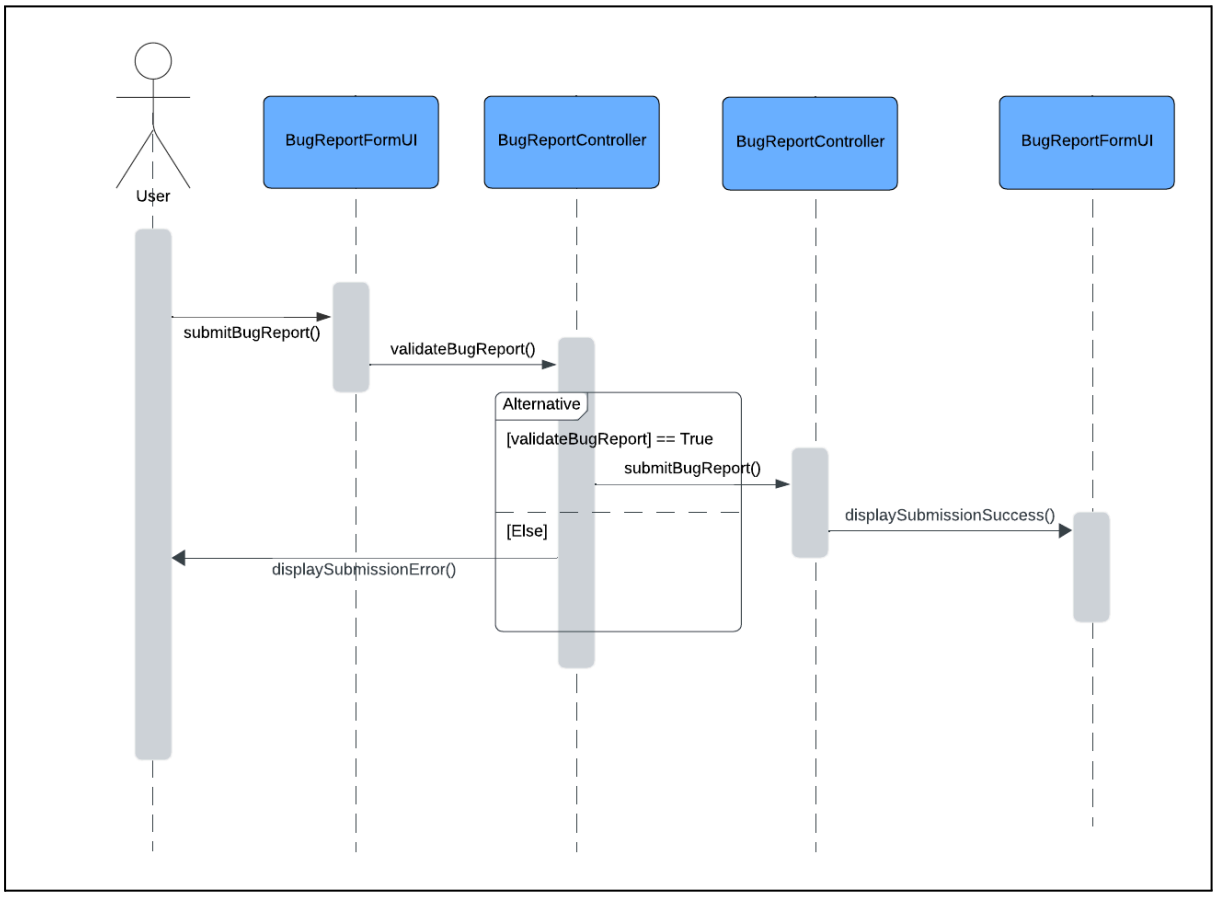
5.4. changePreference



5.5. Logout



5.6. reportBug



6. Initial Dialog map