Software Requirements Specification

for

Urban

Version 2.0 approved

Prepared by:
Kan Huai Feng, Kai (U2121822L)
Goh Soong Wen, Ryan (U2120980L)
Lee Wei Xian (U2121018E)
Arun Ezekiel (U2122402H)
Anapana Dinesh Kumar (U2122938K)

Tutorial Group: Z59

13/04/2023

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Lee Wei Xian	30/01/23	Added Use Cases, Functional and Non Functional Requirements and Data Dictionary	1.1
Arun Ezekiel	01/02/23	Updated Use Cases	1.2
Lee Wei Xian	16/02/23	Added Dialog Map, UML, Sequence Diagram, BCE Diagram	1.3
Goh Soong Wen, Ryan	22/03/23	Updated UML Diagram, Use Case Diagram, Use Case Description and Sequence Diagram	1.4
Kan Huai Feng, Kai	03/04/23	Added all components of introduction and overall descriptions.	1.5
Kan Huai Feng, Kai	04/04/23	Added and updated 3.2, 3.3, 3.4 and all components of System Features and other Nonfunctional Requirements.	1.6
Kan Huai Feng, Kai	06/04/23	Added Appendix D: Black Box Testing and all components of it.	1.7
Kan Huai Feng, Kai	07/04/23	Added Appendix C: White Box testing and all components of it.	1.8
Kan Huai Feng, Kai	08/04/23	Added Appendix A, updated System Features and Other Non-Functional Requirements	1.9
Kan Huai Feng, Kai	11/04/23	Updated 3.1 and System Features	2.0

1. Introduction

1.1 Purpose

The document presented outlines the software requirements for the "Urban" application. It encompasses both the functional and non-functional aspects of the app, including the user interface, design, analysis, implementation, and constraints.

1.2 Document Conventions

Title: Times New Roman, Size 18, Bold
Sub-Title: Times New Roman, Size 14, Bold
Body: Times New Roman, Size 12

1.3 Intended Audience and Reading Suggestions

The document is intended for project developers, managers, professors, teaching assistants and general users.

It should be understood by future developers who are interested in creating new features, adaptation or maintenance.

It is recommended that the document be read in the order presented on the content page.

1.4 Product Scope

Urban is a mobile application designed to cater to the fitness and well-being of its users. With its powerful GPS tracking feature, Urban allows users to monitor and analyse their running or cycling sessions with precision and ease. The application provides detailed reports on users' progress, including historical data and goal tracking to help them stay motivated and on track with their fitness objectives.

The benefits of Urban are vast and wide-ranging, offering users a comprehensive solution to their fitness tracking needs. The app's sophisticated design and user-friendly interface ensure that users can navigate seamlessly through its features and functions, making the tracking experience as smooth and effortless as possible.

1.5 References

As depicted, Urban consults the pertinent API documentation for references.

1. MongoDB - https://www.mongodb.com/docs/
2. React Native Expo - https://docs.expo.dev/

2. Overall Description

2.1 Product Perspective

Urban is an application that enhances several current product families. By utilizing GoogleMaps APIs, Urban provides precise tracking and visualization of the map, while also personalizing the user experience with MongoDB by storing and displaying past cycling sessions and goals. Urban is developed using React Native and Expo.

2.2 Product Functions

Major Functions:
- Login
- Register for an Urban account
- Display Map
- Search for destination and find route
- Track Cycling/Running Sessions
- View Session History

Minor Functions:
- Set and Edit Goals
- Manage Profile
- Pause/Resume live session

2.3 User Classes and Characteristics

Beginner Cyclists/Runners – This class of users will have low frequency of use. They may want to start tracking their running/cycling sessions. They will use the app to search for recommended routes to their destination of choice.

Leisure Bikers/Runners – This class of users will have moderate frequency of use. They will use the app to keep track of their sessions and to find routes to their destinations.

Developers – This class of users will utilize the system to uphold the functionality and ensure that the app remains current.

2.4 Operating Environment

Due to its Progressive Web Application (PWA) nature, Urban is compatible with any mobile devices. It is a type of web application that uses modern web technologies to provide users with an experience similar to that of a native mobile application. PWAs are designed to be fast, reliable, and engaging, with features such as offline access, push notifications, and the ability to be installed on the user's device. The application can be downloaded on any app store.

Urban uses the following frameworks:
- React Native 18.2.0
- Expo 48.0.9
- Yarn 1.22.19
- NodeJS 18.15.0

- MongoDB 6.0

Urban’s UI is built using React Native. React Native is an open-source mobile application development framework created by Facebook. It allows developers to build mobile applications for iOS, Android, and other platforms using JavaScript and the React framework. It provides a fast, efficient, and flexible way to develop mobile user interfaces using JavaScript and a declarative syntax. Its pre-built components, virtual DOM, and style sheet make UI development more straightforward and less time-consuming.

Expo is a set of tools and services for building, deploying, and managing mobile applications using React Native. It includes a development environment, build tools, and a library of pre-built components and APIs that simplify the development of React Native applications.

Yarn is a package manager for JavaScript that was created by Facebook. Yarn allows developers to manage dependencies more efficiently, by caching packages and performing installs in parallel. It also provides a lockfile mechanism that ensures consistent installations across different environments.

NodeJS is used to create the development environment. It is used to install dependencies that the app needs.

MongoDB is used to store data. MongoDB is a cross-platform, document-oriented NoSQL database program that uses JSON-like documents with optional schemas. It is designed to provide high performance, scalability, and flexibility, making it a popular choice for building modern web applications.

2.5 Design and Implementation Constraints

Urban is only available in English. As the client will oversee maintaining and developing new features, a constraint will be the client’s knowledge in JavaScript, React Native and MongoDB.

2.6 User Documentation

Urban is designed to be easy to use and user friendly. A demo video is provided. Proceed to the following link for more information.

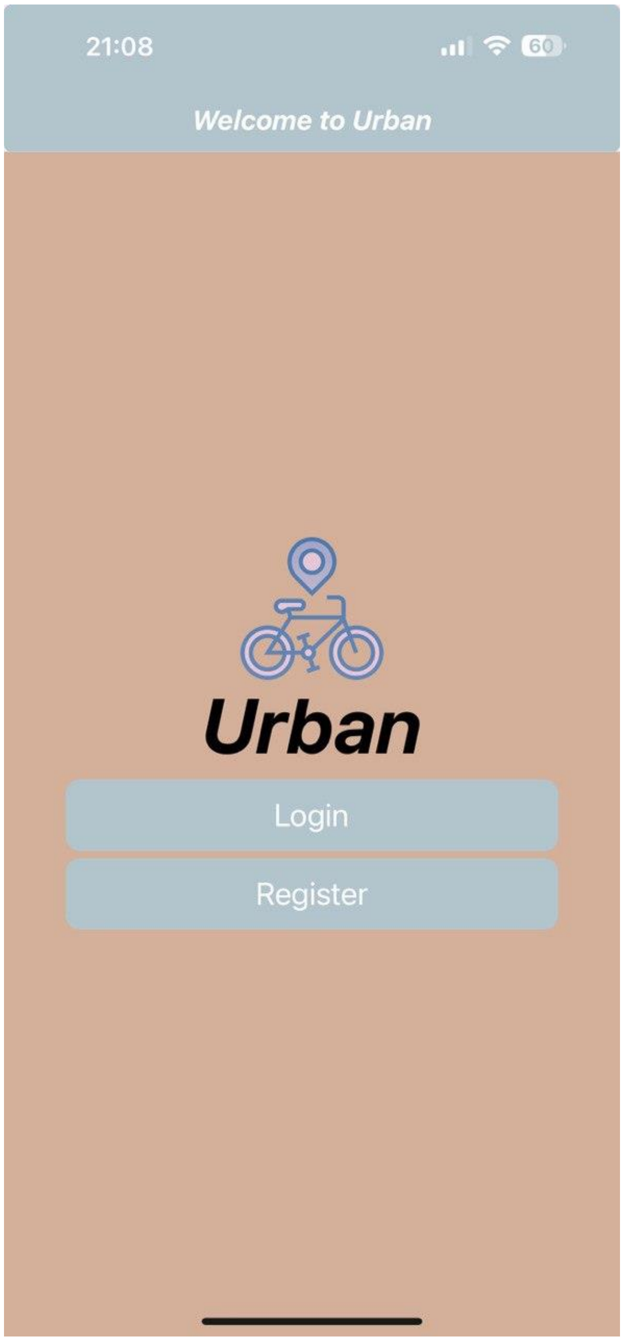
https://drive.google.com/drive/folders/1rJOHo8IEsWW-M3iDzBEaCOcLKzpcfIp8

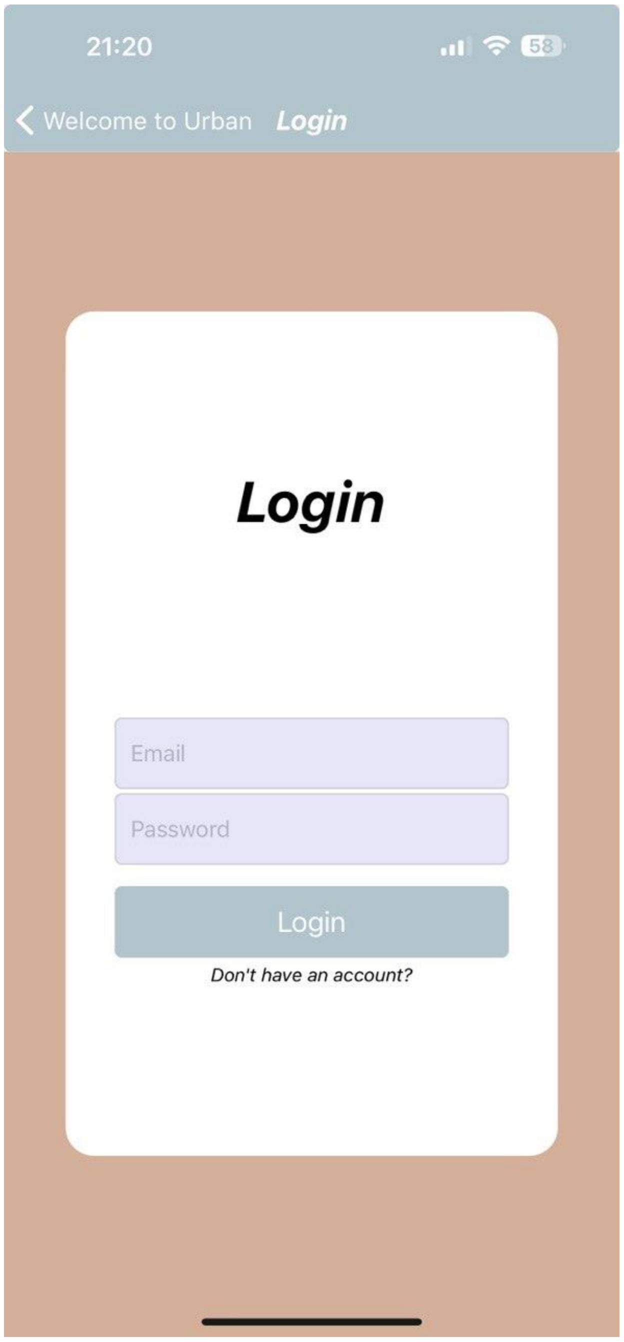
2.7 Assumptions and Dependencies

The team assumes that users are connected to the internet and that they have their location services enabled.

3. External Interface Requirements

3.1 User Interfaces

 21:08 60 Welcome to Urban Urban Login Register	**Urban Main Page** Users can choose to <Login> or <Register> from this page. The page is designed to provide a secure and convenient way for users to access the application and to create a new account if necessary.

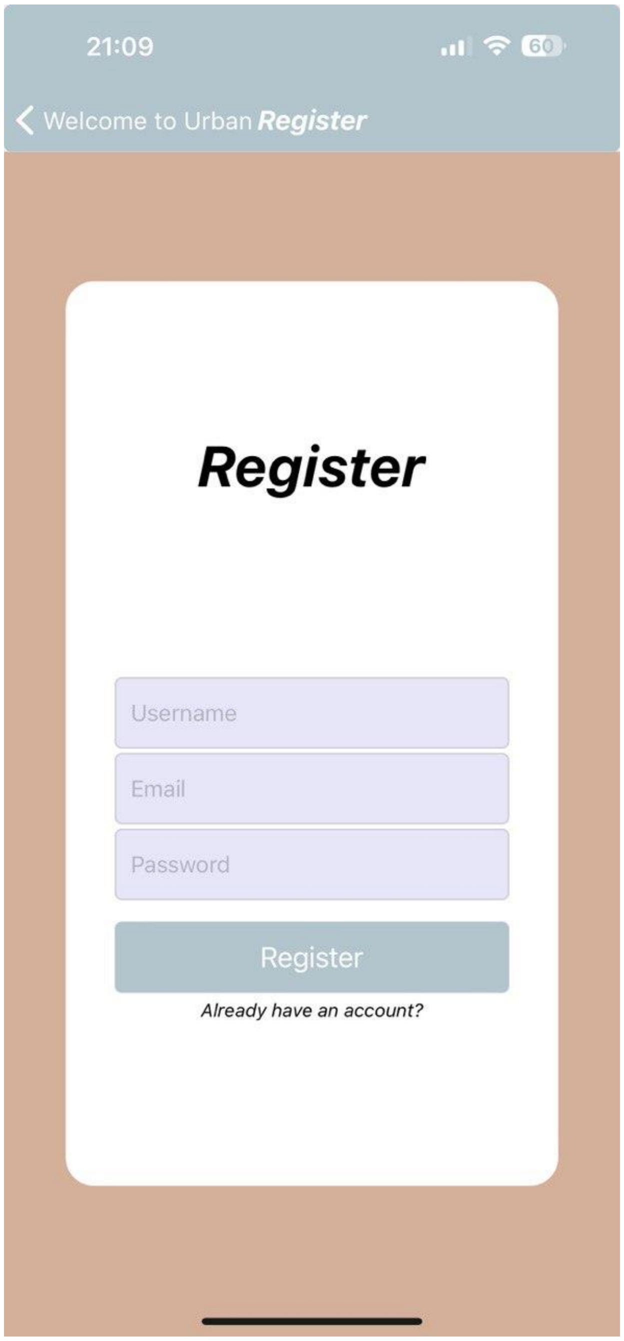


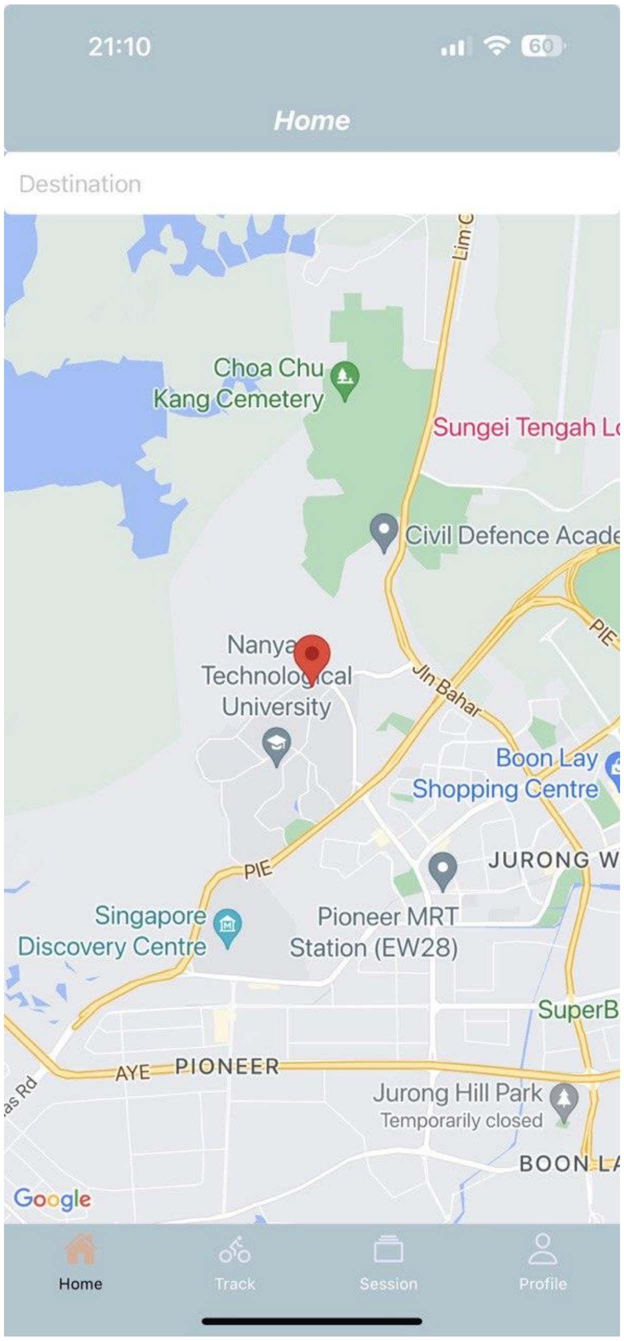
Login Page

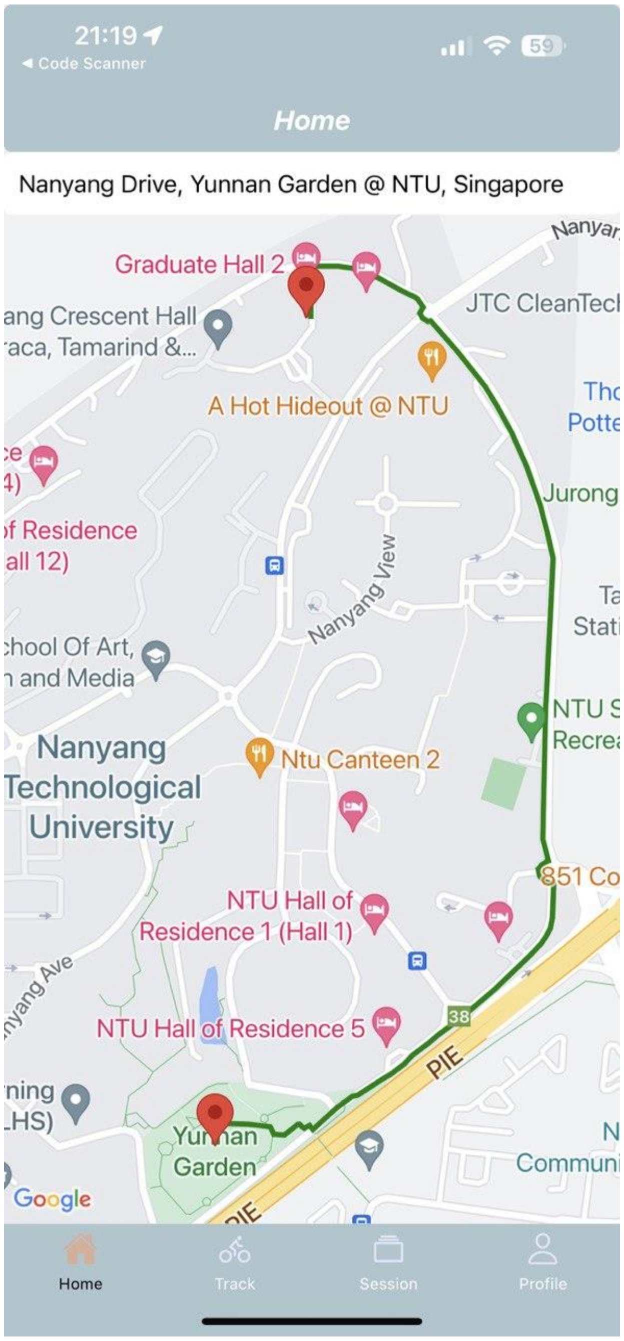
Users are prompted to enter their email address and password in order to gain access to their account. The login form is designed to be simple and straightforward, with clear instructions and intuitive input fields that make the login process a breeze.

In order to ensure that only authorized users are granted access to their account, the login details are encrypted and authenticated.

Once the user has entered their login credentials, the application verifies the authenticity of the provided information. If the credentials are valid, the user is granted access to their account and can start exploring the application's features and functionalities. On the other hand, if the credentials are invalid, the application prompts the user to re-enter their details or reset their password.

 21:09 Welcome to Urban Register Register Username Email Password Register Already have an account?	**Register Page** Users can register for a new Urban account on this page. The registration form is prominently displayed, with clear instructions and intuitive input fields that guide users through the process of creating their account. To create a new account, users are required to provide some basic information such as their desired username, email address, and password. Once the user has completed the registration process and their account is successfully created, they are redirected to the login page.

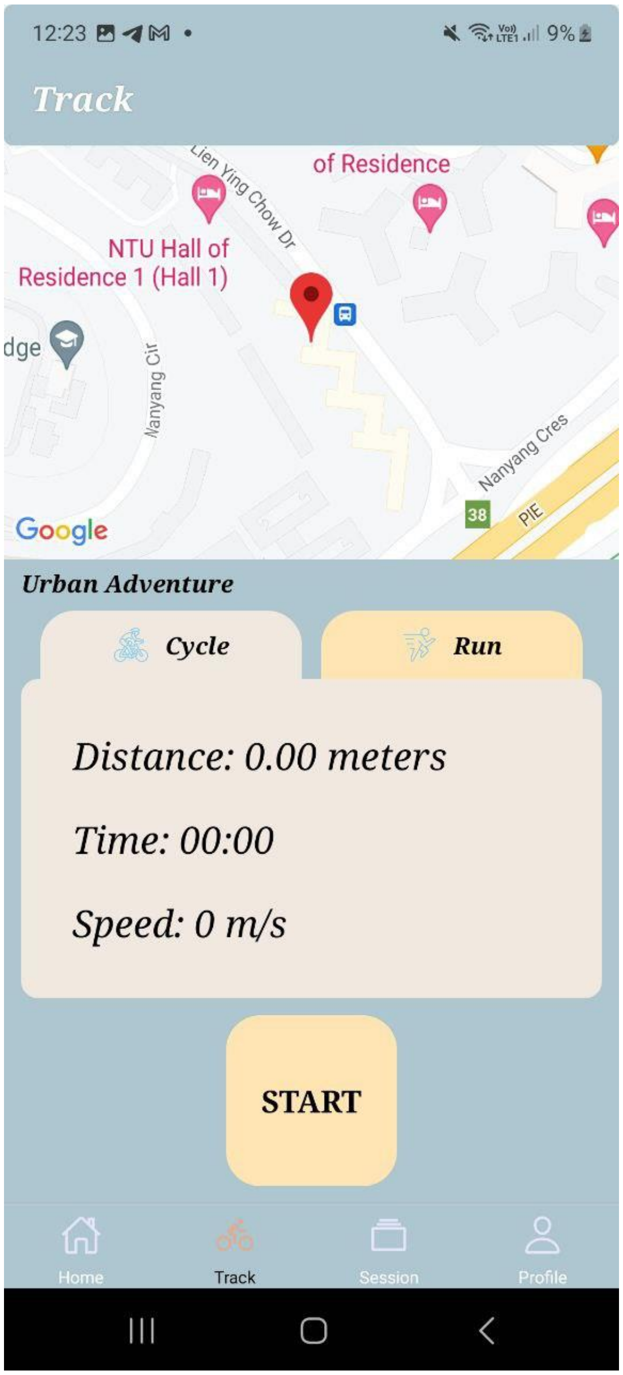
 21:10 Home Destination Choa Chu Kang Cemetery Sungei Tengah L Civil Defence Acade Nanyang Technological University Jln Bahar Boon Lay Shopping Centre JURONG W PIE Singapore Discovery Centre Pioneer MRT Station (EW28) SuperB AYE PIONEER Jurong Hill Park Temporarily closed BOON LA Google Home Track Session Profile	**Urban's Home Page** The home page of the application is designed to provide the user with an intuitive and immersive experience. It features a dynamic map that displays the user's current location, making it easy for them to navigate and explore the surrounding area. The map is interactive, allowing the user to zoom in and out and move around to explore different locations. At the bottom of the home page, a navigation bar is prominently displayed, providing quick access to the application's key features and functions. This navigation bar is consistently visible throughout all other pages of the application, ensuring that the user can easily access the features they need, regardless of where they are in the application. One of the standout features of the application is its ability to help users find a destination quickly and easily. By simply entering a desired destination into the search bar, the application will automatically find the best route to the place of interest. This feature is powered by advanced algorithms and real-time traffic data, ensuring that the user always receives the most accurate and up-to-date directions.

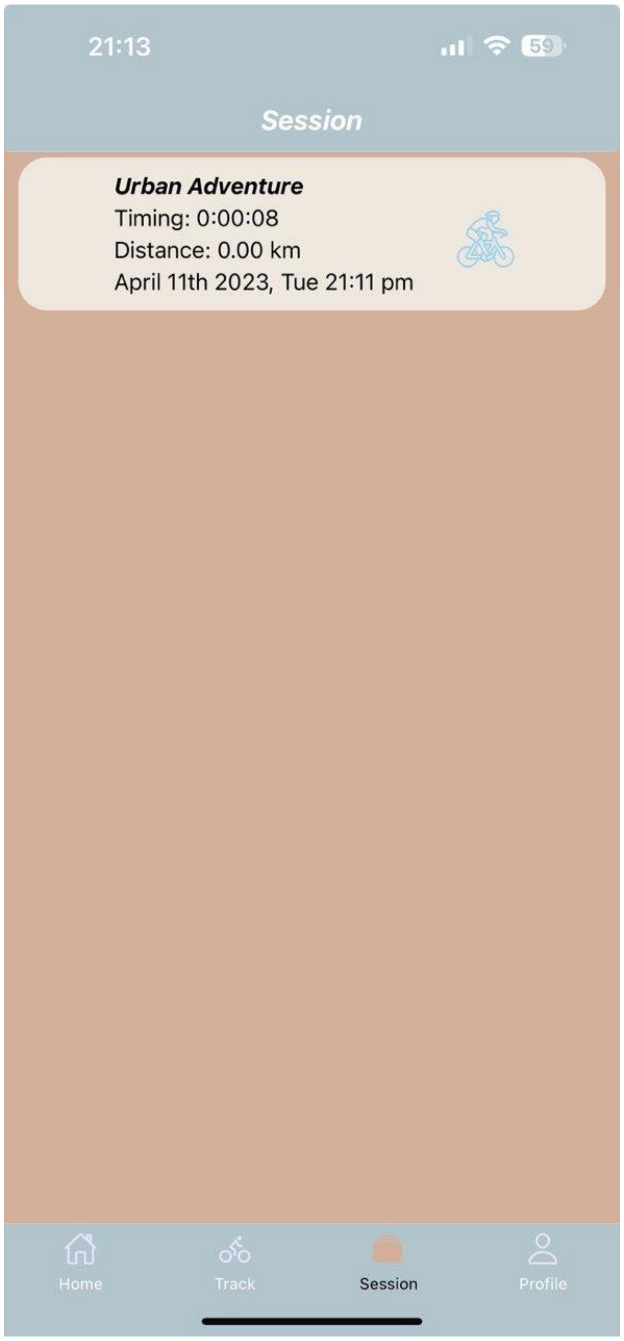
	

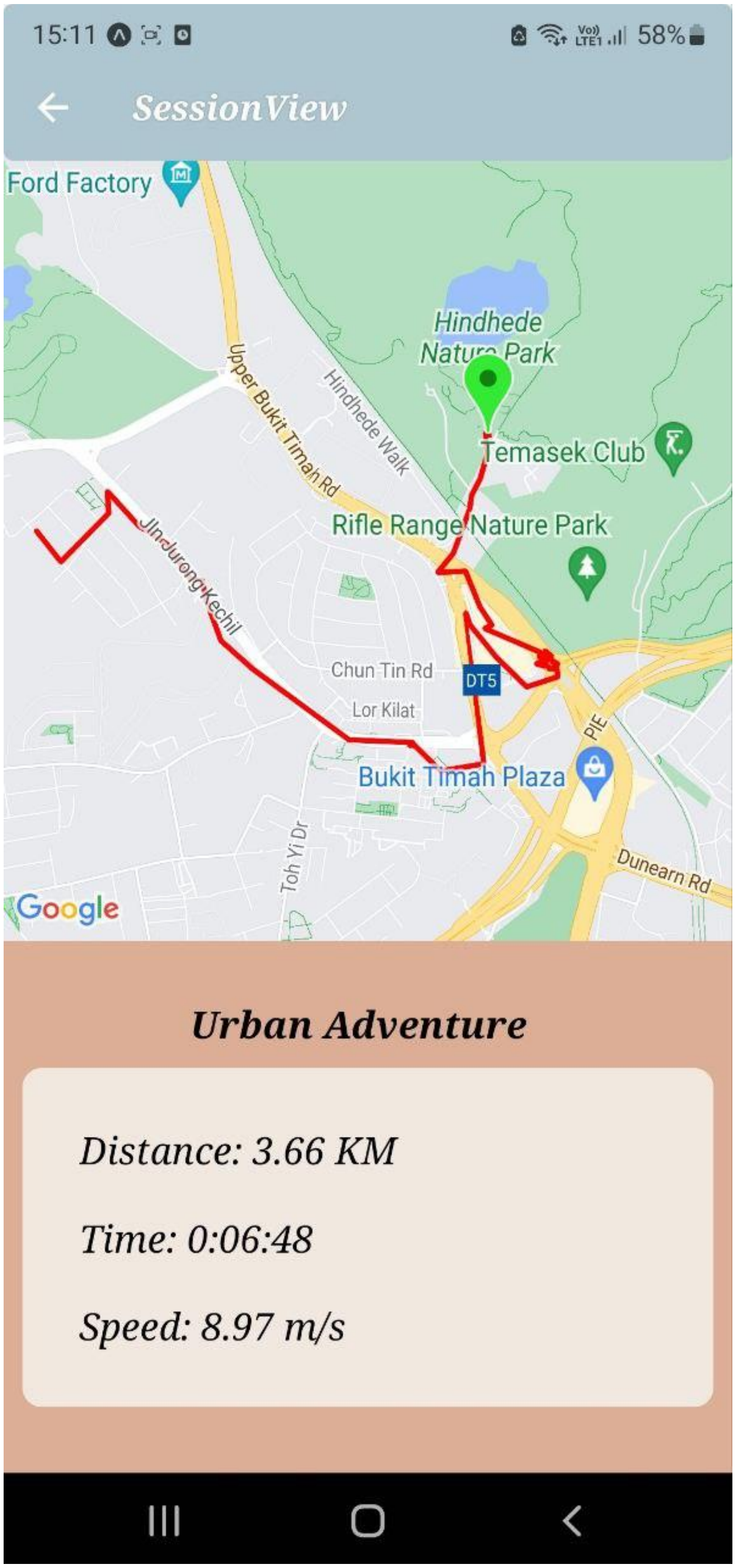
21:19
◀ Code Scanner

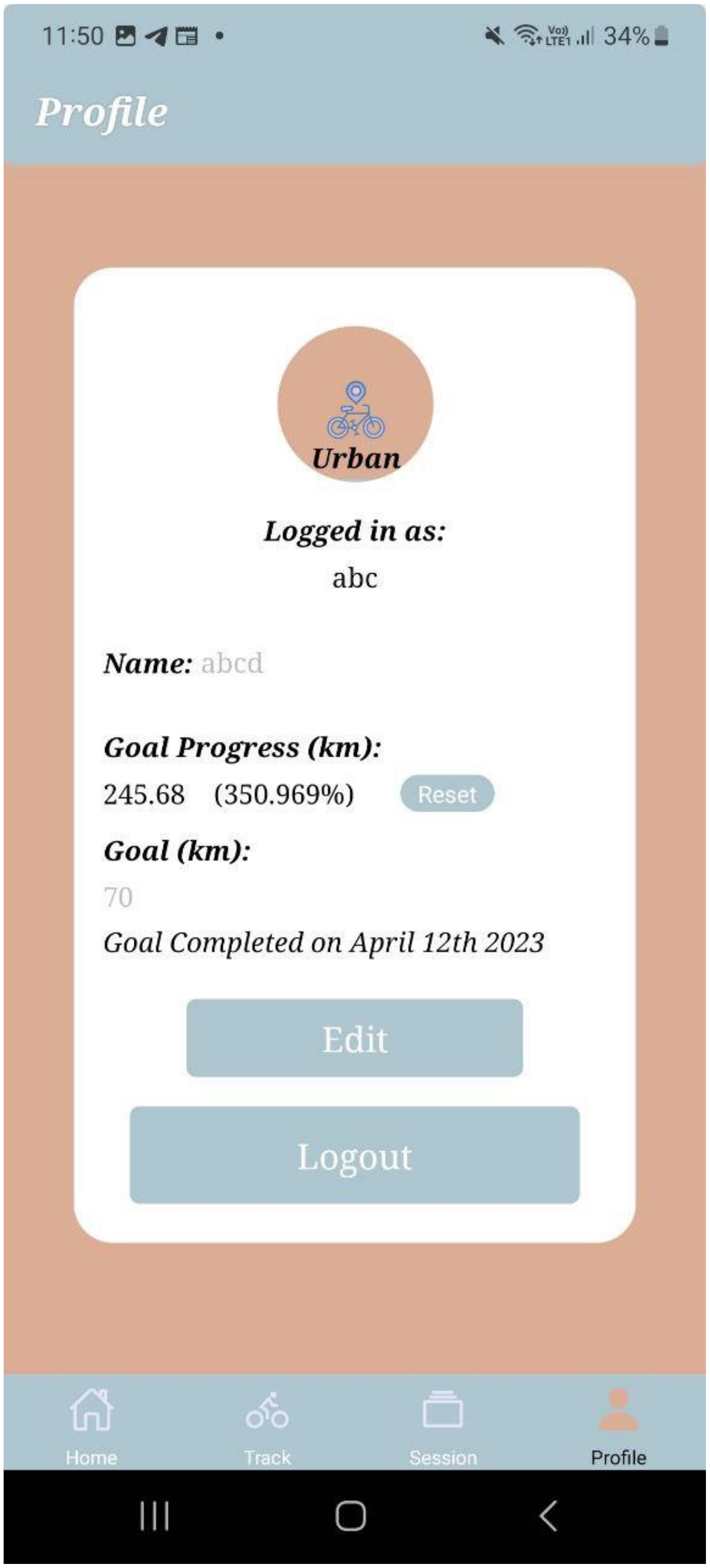
Home

Nanyang Drive, Yunnan Garden @ NTU, Singapore

 12:23 9% Track of Residence Lien Ying Chow Dr NTU Hall of Residence 1 (Hall 1) dge Nanyang Cir Nanyang Cres 38 PIE Google Cycle Run START Home Track Session Profile Urban Adventure Distance: 0.00 meters Time: 00:00 Speed: 0 m/s	**Tracking Page** The tracking page features a large, easy-to-read display that shows the user's current distance, time, and speed. The user can easily start, pause, or stop their session by pressing the corresponding buttons on the screen. This makes it easy for the user to take a break or adjust their pace without interrupting their progress. The user can also rename the title of the session in this page. As the user runs, the application automatically saves their session data in the background. This means that even if the user accidentally closes the application or loses their internet connection, their progress is saved and can be retrieved later.

 21:13 Session **Urban Adventure** Timing: 0:00:08 Distance: 0.00 km April 11th 2023, Tue 21:11 pm	**Session Page** Users can track and monitor their progress over time on this page. It provides users with an easy-to-use interface for reviewing their previous sessions, including details such as the session title, total time taken, total distance run or cycled, date of session, and end time. The page shows a list of their previous sessions. Each session is represented by an icon of either a cyclist or a runner, specific to the type of session it was. When the user clicks into a specific session, they are presented with a detailed view that shows the route taken, total distance, total time, and average speed during the session. Users can use the session page to track their progress over time, identify areas where they need to improve, and set new goals for themselves.

 15:11 58% *SessionView* **Urban Adventure** Distance: 3.66 KM Time: 0:06:48 Speed: 8.97 m/s	

 11:50 34% **Profile** Urban **Logged in as:** abc **Name:** abcd **Goal Progress (km):** 245.68 (350.969%) Reset **Goal (km):** 70 *Goal Completed on April 12th 2023* Edit Logout Home Track Session Profile	**Manage Profile Page** This page provides users with an easy-to-use interface for managing their personal information, including their profile picture, email, username, and fitness goals. Users can view their current progress towards their fitness goals, including the distance they have covered in kilometers and the percentage of their goal that they have achieved. If users want to edit their goals or username, they can do so easily by clicking on the <Edit> button. This allows them to update their fitness goals and personal information as needed, ensuring that their profile remains up-to-date and accurate. The profile management page also provides users with the ability to upload or change their profile picture. This allows them to personalize their experience whilst using the application. If users wish to reset their progress, they can do so by clicking the <Reset> button.

3.2 Hardware Interfaces

Urban utilizes the various hardware components inherent in mobile devices to enhance the overall user experience. The app relies on sensors like the accelerometer and geomagnetic field to determine geolocation and orientation during sessions. Additionally, wireless connectivity is required to access online databases and APIs, enabling seamless data transfer and an uninterrupted user experience.

3.3 Software Interfaces

Urban uses the following APIs:

1) React Native 18.2.0
    a. PWA rendering and client side logic
2) Google Maps API 3.52
    a. Displays Map, Routes, User Location
    b. Tracks user location and utilize for session visualization
3) MongoDB 6.0
    a. Stores User Data
        i. Profile details
        ii. Session details
        iii. User goals
    b. Authentication
        i. Create new user account
        ii. Verify login credentials

3.4 Communications Interfaces

Urban uses MongoDB Wire protocol to communicate with MongoDB servers. It is a binary protocol used to communicated between MongoDB clients and servers.

4. System Features

4.1 Account Registration

4.1.1 Description and Priority

Users must register for an account if they do not possess one to use the functionalities of the application.

Priority: High.

4.1.2 Stimulus/Response Sequences

1. The user presses on the <Register> button on the application's start page.

2. The system displays the application's registration page.
3. The user enters a username, password, and a unique email.
4. The user submits valid username, password, and email by pressing the <Sign Up> button.
5. The server validates the input information and sends a request to MongoDB.
6. MongoDB ensures the email input is unique.
7. MongoDB saves the username, password, and email.
8. The system displays a successful registration message.
9. The system displays the application's home screen.

4.1.3 Functional Requirements

REQ-1: User must be able to create a new user account if they do not possess one at the application's login page.

1.1 Users must input a unique username.

1.1.1 The username must be of string data type containing alphanumeric characters.

1.1.2 The username must have at least 4 characters and at most 20 characters.

1.2 Users must input their password.

1.2.1 The password must meet the following requirements:

1.2.1.1 The password must be of minimum 8 characters and at most 20 characters.

1.2.1.2 The password must include at least 1 capital letter (A-Z).

1.2.1.3 The password must include at least 1 number (0-9).

1.2.2 If the password does not meet all the requirements of 1.2.1., the application will prompt a message: "Password must be between 8-20 characters long!!!" or "At least 1 capital [A-Z] and 1 number [0-9]!!!".

1.3 Users must input their email address.

1.3.1 The email address must be of string data type.

1.3.2 The email address must include "@" and ".".

1.3.3 If the email address is already registered, the application will prompt a message: "Email Taken".

1.4 The application must allow users to register for an account upon pressing the **<Sign Up>** button.

1.4.1 If there are input fields left blank, the application will prompt a message: “Username/Email/Password cannot be Empty!!!”.

1.4.2 If conditions 1.1. to 1.3. are met and there are no empty input fields, the application must store the user’s login details into the cloud database.

1.4.2.1 The user must be given a default profile picture.

1.4.2.2 The application will prompt a message: “Account successfully registered.”.

1.4.2.3 The application must redirect users to the application’s home page.

4.2 Login

4.2.1 Description and Priority

The user indicates intent to log in to his previously registered account with his registered email and password to access the functionalities of the application.

Priority: High.

4.2.2 Stimulus/Response Sequence

1. The user presses on the <Login> button on the application’s start page.
2. The system displays the application’s login page.
3. The user enters a registered email and the associated password.
4. The user submits the email and password by pressing the <Login> button and sends a request to MongoDB.
5. MongoDB verifies the email and password.
6. The system displays the application’s home screen.

4.2.3 Functional Requirements

REQ-2: Users must be able to log into their accounts which they have previously registered by inputting a registered email and its associated password at the application’s login page.

2.1 The application must check the validity of the email and password.

2.1.1 If the email and password are valid, the user will be directed to the application’s home page.

2.1.1 If either the email or password is invalid, the application will prompt a message: "Invalid Email" or "Invalid Password"

4.3 Edit Profile

4.3.1 Description and Priority

The user indicates interest to edit his profile. The profile will display his username, first name, last name, email, total distance to date and total hours to date on sessions.

Priority: High.

4.3.2 Stimulus/Response Sequence

1. The user can make changes to his profile by pressing the <Edit> icon on the application's profile page.
2. Users can reset their goal process by pressing the <Reset> icon on the application's profile page.
3. The system displays the application's edit profile page.
4. Users can update his profile by inputting a new username.
5. Users can update his profile picture by uploading a new picture.
6. Users can save his changes by pressing the <Save> button.
7. The server sends a request to MongoDB.
8. MongoDB will update the user profile information.

4.3.3 Functional Requirements

REQ-3: Users must be able to manage their personal details at the application's profile page.

3.1 The application's profile page must display the following information about the user:

3.1.1 Profile Picture.

3.1.1.1 Default Picture is given.

3.1.2 Username.

3.1.3 Email.

3.1.4 Goal Progress

3.1.5 Goal and Goal Completion Status

3.1.5.1 Default value for Goal is given

3.2 Users must be able to edit their account details at the application's edit profile page.

3.2.1 Users must be able to edit their profile picture.

3.2.2 Users must be able to edit their username.

3.2.2.1 Username provided must meet condition 1.1.

3.3 The application must allow users to save their edited account details.

3.3.1 The application must update the user's account details on MongoDB.

3.4 The application must allow users to reset their goal progress.

3.4.1 The application will prompt the user to confirm the decision to reset the goal progress.

3.5 The application must allow users to cancel the current changes to their account details.

4.4 Manage Tracking

4.4.1 Description and Priority

The user indicates intent to record a session. The user can start, stop, pause and resume a session. The session will show the user's current session details, including his current location on the map, the route from the initial start point of the session to the user's current location on the map, total distance travelled, time taken, and average speed.

Priority: High.

4.4.2 Stimulus/Response Sequence

1. The user can select the type of session by pressing either the <Cycle> or <Run> icon.
2. The user can change the session name by pressing <Urban Adventure>.
3. The user starts a session by either pressing the <Start> icon.
4. The system will start recording the session.
5. The system will constantly update the current session's details.
6. The user stops the current session by pressing the <Stop> button.
7. The system will stop recording the session and send a request to MongoDB.
8. MongoDB will save the session with the user's id and session's details.
9. The system displays the session details.

4.4.3 Functional Requirements

REQ-4: Users must be able to track their current session on the application's session page.

4.1 The application's session page must display and continuous update the following information:

4.1.1 The user’s current location on the map.

4.1.1.1 Default value is given if location permission is not given.

4.1.2 Total distance traveled (in kilometers, to 2 decimal points of accuracy) during the current session.

4.1.3 Time taken (in HH:mm:ss format) since start of current session.

4.1.4 The average speed (in meters per second, to 2 decimal points of accuracy) of the current session.

4.2 A pre-set value will be assigned as the default for the session title and session type if the user does not make any changes.

4.3 Users must be able to pause their current session.

4.4 Users must be able to resume a paused current session.

4.5 Users must be able to stop a current session.

4.5.1 The session must be added to the user’s history.

4.5.1.1 An alert message “Session successfully saved” will be shown.

4.5 View Session

4.5.1 Description and Priority

The user indicates the intent to view details about his past sessions. The details include the name of session, the type of session, date, start time, distance travelled, timing, average speed and cycling route of the session.

Priority: High

4.5.2 Stimulus/Response Sequence

1. The user can view his past sessions by pressing the <Sessions> button on the application’s home page.
2. The system displays the application’s past sessions which were saved in MongoDB.
3. The user can press on the sessions to view past session details.
4. The system will display the date, end time, distance travelled, timing and route of the session.

4.5.3 Functional Requirements

REQ-5: Users must be able to view the details of their past sessions at the application's session history page.

5.1 The application must sort the past sessions by recency (from newest to oldest session).

5.2 The application must display the following details of past sessions on the application's session history page:

5.2.1 The date of the session (in month/day/year).

5.2.2 The end time of the session (in 24 hour format).

5.2.3 Distance traveled (in kilometers, to two decimal points of accuracy).

5.2.4 The total timing of the session (in HH:mm:ss).

5.2.5 The name of the session.

5.2.6 The type of the session.

5.3 The user must be able to click on the session to view the following additional information:

5.3.1 The route, traced on a map.

5.3.2 The average speed (in m/s).

4.6 Edit Goal

4.6.1 Description and Priority

The goal will be displayed on the top of the application's session page with a default goal of 50 kilometers. The goal counter will start with a value of 0 and is the sum of distance travelled during the sessions recorded in the current month. The user indicates the intent to set a new goal.

Priority: High

4.6.2 Stimulus/Response Sequence

1. The user can edit a goal by pressing the <Edit> icon on the application's sessions page.
2. The user inputs the goal (in kilometers).
3. The user submits the goal by pressing the <Confirm> button.
4. The system validates the goal and sends a request to MongoDB.
5. The user goal is updated on the MongoDB.
6. The system successfully displays the new goal.

4.6.3 Functional Requirements

REQ-6: The application must allow users to set goals at the application's session history page.

6.1 Users must input the following information:

6.1.1 Goal (in kilometers, to zero decimal points of accuracy).

6.1.1.1 The goal value is between 10 - 999.

6.2 The user goal must display the following information:

6.2.1 Goal (in kilometers, to zero decimal points of accuracy).

6.2.2 Current total distance of sessions for the month (in kilometers, to two decimal points of accuracy).

6.2.3 Percentage of completion (in percent, to 3 decimal points of accuracy)

6.2.3.1 When completed, the system shows the date of completion in Month/Day/Year format.

6.3 Users must be able to edit their goal.

6.3.1 User will not be able to edit their goal to lower than their current distance completed.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 The application requires a device that supports a stable network connection.
5.1.2 The application must be able to continue a session in the background.
5.1.3 The application must be able to track the user's location in the background.
5.1.4 If network connection is lost, the application must be able to continue updating the session once network connection is up.
5.1.5 The application must meet or exceed 99% uptime.
5.1.6 80% of users must be able to start recording a session in less than 3 seconds.
5.1.7 80% of users must be able to login in less than 3 seconds.

5.2 Safety Requirements

5.2.1 The application must not prompt user interaction, including notifications, during a session.

5.3 Security Requirements

5.3.1 Data must be protected in accordance with PDPA.
5.3.2 Users must not be able to see the following information about other users:
5.3.2.1 Email
5.3.2.2 Password

5.4 Usability Requirements

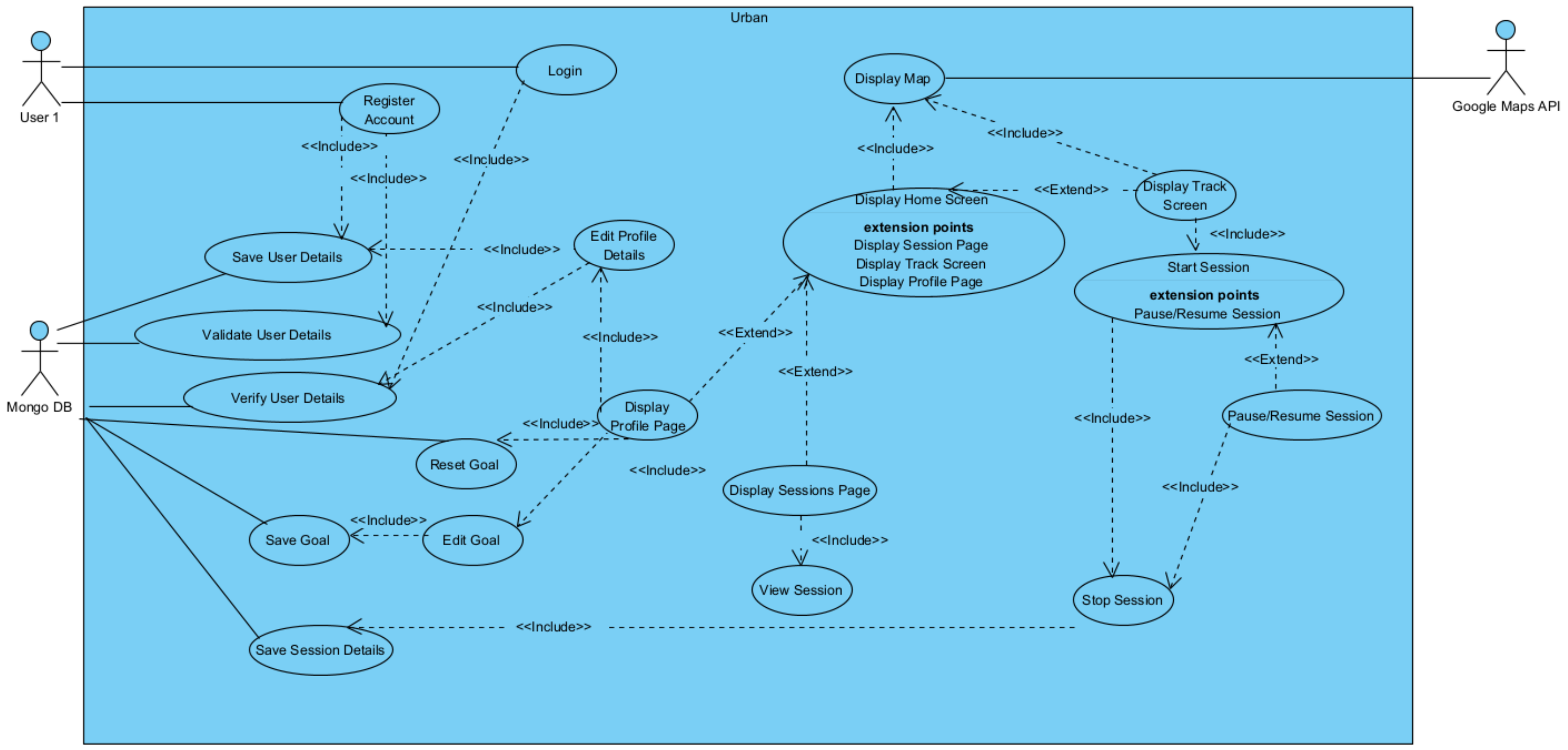
5.4.1 The application must be available in English.
5.4.2 The application must notify users of new updates when new updates are available.
5.4.3 The application must give first-time users a quick and intuitive tutorial.
5.4.3.1 Each instruction must not exceed 20 words.
5.4.3.2 90% of users must be able to complete the tutorial in less than 2 minutes.
5.4.4 The map display in 4.4 will make use of Google Maps API.
5.4.5 The application must display up-to-date information about the user.

Appendix A: Glossary

Term	**Definition**
System	The system manages the functionalities of the application.
User	A user is an individual who is using the application.
Server	The server manages the remote functionalities of the application.
Account	An account allows users to access the features of the application. All accounts will have a profile.
Profile	A data structure of a user’s personal information. This refers to profile picture, username, first name, last name, email, total distance travelled and total hours travelled.
Session	A session is “started”, “paused” and “ended” by a user. It is used to capture the user’s current location, the route from the initial start point of the user to the user’s current location on the map, total distance travelled during the current session, time taken during the current session and the average speed.
Cloud Database	The Cloud database helps organise, store, and manage data, built on a public or hybrid cloud environment.
APIs	Application Programming Interfaces are intermediaries of software that allow two or more different applications to communicate with each other and share data.

Appendix B: Analysis Models

B.1 Use Case Diagram



B.2 Use Case Descriptions

Use Case ID:	UC001		
Use Case Name:	Account Registration		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	User, MongoDB
Description:	Users must register for an account if they do not possess one to use the functionalities of the application.
Preconditions:	User does not have a valid account and wants to register for one. The user must be on the application's start page.
Postconditions:	Account is created in MongoDB with input details. The user will be redirected to the application's home page.
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. The user presses on the <**Register**> button on the application's start page. 2. The system displays the application's registration page. 3. The user enters a username, password, and a unique email. 4. The user submits valid username, password, and email by pressing the <**Sign Up**> button. 5. The server validates the input information and sends a request to MongoDB. 6. MongoDB ensures the email input is unique. 7. MongoDB saves the username, password, and email. 8. The system displays a successful registration message. 9. The system displays the application's home screen. 10. The use case ends.
Alternative Flows:	UC001.AC.1 Invalid username, password, or email 1. The system prompts the user for the invalid inputs. 2. Use case resumes at main flow step 3. UC001.AC.2 Email taken 1. The system displays an "Email registered" message. 2. The system prompts the user for an unregistered email. 3. Use case resumes at main flow step 3.
Exceptions:	UC001.EX.1 Cancellation of User Registration 1. The user aborts the use case by pressing on the <**Back>** button. 2. The system does not save any details. 3. The user will be redirected to the application's start page. 4. The use case ends.
Includes:	SVR.UC001, SVR.UC002
Special Requirements:	
Assumptions:	When the user enters the application's registration page, the user must successfully create an account.
Notes and Issues:	

Use Case ID:	UC002		
Use Case Name:	Login Account		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	User, MongoDB
Description:	The user indicates intent to log in to his previously registered account with his registered email and password to access the functionalities of the application.
Preconditions:	User has previously created a valid account (stored in MongoDB). User is on the application's start page.
Postconditions:	User successfully logs in. The user is redirected to the application's home screen.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The user presses on the <**Login**> button on the application's start page. 2. The system displays the application's login page. 3. The user enters a registered email and the associated password. 4. The user submits the email and password by pressing the <**Login**> button and sends a request to MongoDB. 5. MongoDB verifies the email and password. 6. The system displays the application's home screen. 7. The use case ends.
Alternative Flows:	UC002.AC.01 Missing email and/or password 1. The system prompts for missing email and/or password. 2. The use case resumes at main flow step 3. UC002.AC.02 Incorrect email and/or password 1. The system displays an "Incorrect email and/or password" message. 2. The system prompts for an email and password. 3. The use case resumes at main flow step 3.
Exceptions:	UC002.EX.1 Cancellation of Account Login 1. The user aborts the use case by pressing on the <**Back**> button. 2. The system displays the application's start page. 3. The use case ends.
Includes:	SVR.UC003
Special Requirements:	
Assumptions:	When the user enters the application's account login page, the user must successfully login to their registered account.
Notes and Issues:	

Use Case ID:	UC003		
Use Case Name:	Manage Profile		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	User, MongoDB
Description:	The user indicates interest to view his profile. The profile will display his username, email, total distance to date, profile picture, goal, goal progress and goal completion progress. The user can edit his profile details.
Preconditions:	User is logged in. User is on the application's home page.
Postconditions:	The user's information is updated on MongoDB.
Priority:	Mid
Frequency of Use:	Mid
Flow of Events:	1. The user presses the <**Profile**> button on the application's home page. 2. The system displays the application's profile page. 3. The use case ends.
Alternative Flows:	UC003.AC.01 Edit profile 1. The user can make changes to his profile by pressing the <**Edit**> icon on the application's profile page. 2. Users can reset their goal process by pressing the <**Reset**> icon on the application's profile page. 3. The system displays the application's edit profile page. 4. Users can update his profile by inputting a new username. 5. Users can update his profile picture by uploading a new picture. 6. Users can save his changes by pressing the <**Save**> button. 7. The server sends a request to MongoDB. 8. MongoDB will update the user profile information. 9. The use case resumes at main flow step 2. UC003.AC.02 Cancel edit 1. The user can revert any changes to his profile during current edit by pressing the <**Cancel**> button on the application's edit profile page. 2. The use case resumes at main flow step 2.
Exceptions:	UC003.EX.01 Invalid username 1. The system will display a "Invalid Username" message. 2. The system will prompt the user to enter a new username. 3. The use case resumes at alt flow UC003.AC.01 step 3.
Includes:	SVR.UC002
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC004		
Use Case Name:	Manage Tracking		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	User, MongoDB, Google Maps API
Description:	The user indicates intent to record a session. The user can start, stop, pause and resume a session. The session will show the user's current session details, including his current location on the map, the route from the initial start point of the session to the user's current location on the map, total distance travelled, time taken, and average speed.
Preconditions:	User is logged in. User is on the application's home page.
Postconditions:	The session is stopped and the user's session details are stored on the MongoDB.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The user can select the type of session by pressing either the <**Cycle**> or <**Run**> icon. 2. The user can change the session name by pressing <**Urban Adventure**>. 3. The user starts a session by either pressing the <**Start**> icon. 4. The system will start recording the session. 5. The system will constantly update the current session's details. 6. The user stops the current session by pressing the <**Stop**> button. 7. The system will stop recording the session and send a request to MongoDB. 8. MongoDB will save the session with the user's id and session's details. 9. The system displays the session details. 10. The system displays the application's home page.
Alternative Flows:	UC004.AC.01 Pause and Resume 1. The user can pause a current session by pressing the <**Pause**> button either on main flow step 2 or 3. 2. The system will temporarily stop updating the session. 3. The system will continue updating the session after the user presses the <**Resume>** button. 4. The use case resumes at main flow step 3.
Exceptions:	
Includes:	SVR.UC004
Special Requirements:	
Assumptions:	The user will not <**Pause**> the session indefinitely. The user will eventually press the <**Stop**> button.
Notes and Issues:	

Use Case ID:	UC005		
Use Case Name:	View Session		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	User, MongoDB
Description:	The user indicates the intent to view details about his past sessions. The details include the name of the session, the type of the session, date, start time, distance travelled, timing, average speed and cycling route of the session.
Preconditions:	User is logged in. User has past sessions. User is on the application's home page.
Postconditions:	The system displays the relevant information about the user's past sessions.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The user can view his past sessions by pressing the <**Sessions>** button on the application's home page. 2. The system displays the application's past sessions which were saved in MongoDB. 3. The user can press on the sessions to view past session details. 4. The system will display the date, end time, distance travelled, timing and route of the session. 5. The use case ends.
Alternative Flows:	
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC006		
Use Case Name:	Edit Goal		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	User, MongoDB
Description:	The goal will be displayed on the top of the application's session page with a default goal of 50 kilometres. The goal counter will start with a value of 0 and is the sum of distance travelled during the sessions recorded. The user indicates the intent to set a new goal.
Preconditions:	User is logged in. User is on the application's session page.
Postconditions:	The goal is successfully updated on the MongoDB. The goal is shown on the session page.
Priority:	Low
Frequency of Use:	Mid
Flow of Events:	1. The user can edit the goal by pressing the <**Edit**> icon on the application's sessions page. 2. The user inputs the goal (in kilometres). 3. The user submits the goal by pressing the <**Confirm**> button. 4. The system validates the goal and sends a request to MongoDB. 5. The user goal is updated on the MongoDB. 6. The system successfully displays the new goal. 7. The use case ends.
Alternative Flows:	
Exceptions:	UC006.EX.1 Invalid goal exceeding 999KM 1. The system displays "Goal must not exceed 999KM" message. 2. The system prompts the user to enter a valid goal. 3. The use case resumes at main flow step 2. UC006.EX.2 Cancellation of Goal Setting 1. User aborts the use case by pressing the <**X**> icon. 2. The use case ends.
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	SVR.UC001		
Use Case Name:	Validate User Details		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	Server, MongoDB
Description:	The use case begins when the server receives user details to validate. The use case ends when the input email is unique, while username and password is filled and valid.
Preconditions:	Email, username and password entered.
Postconditions:	Accurately verified the user details and sends an appropriate message to the system.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The server receives user details. 2. The server validates that all user details, including email, username and password, are not missing and are valid. 3. The server sends a request to MongoDB to validate input email. 4. The MongoDB validates the email is not used and in valid format. 5. The server successfully validates all the user details. 6. The use case ends.
Alternative Flows:	SVR.UC001.AC1 Email Taken 1. The server sends an “Email Taken” message to the system. 2. The use case ends. SVR.UC001.AC2 Empty inputs 1. The server sends a “Missing inputs” message to the system. 2. The user case ends.
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	SVR.UC002		
Use Case Name:	Save User details		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	Server, MongoDB
Description:	The use case begins when the actor receives validated user details - email, username and password.
Preconditions:	User details, including email, password and username validated.
Postconditions:	User details are saved successfully on MongoDB.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The server sends a request to MongoDB with validated user details. 2. The server saves the email on MongoDB as a key. 3. The server saves the username and password on MongoDB. 4. The server successfully saved all user details on MongoDB. 5. The use case ends.
Alternative Flows:	
Exceptions:	SVR.UC002.EX1 Storage space full 1. The server aborts the case due to not enough space left in storage. 2. The server sends a "Insufficient Storage, please contact an admin" message to the system. 3. The use case ends.
Includes:	
Special Requirements:	
Assumptions:	The user details will always be successfully saved after validation of details.
Notes and Issues:	

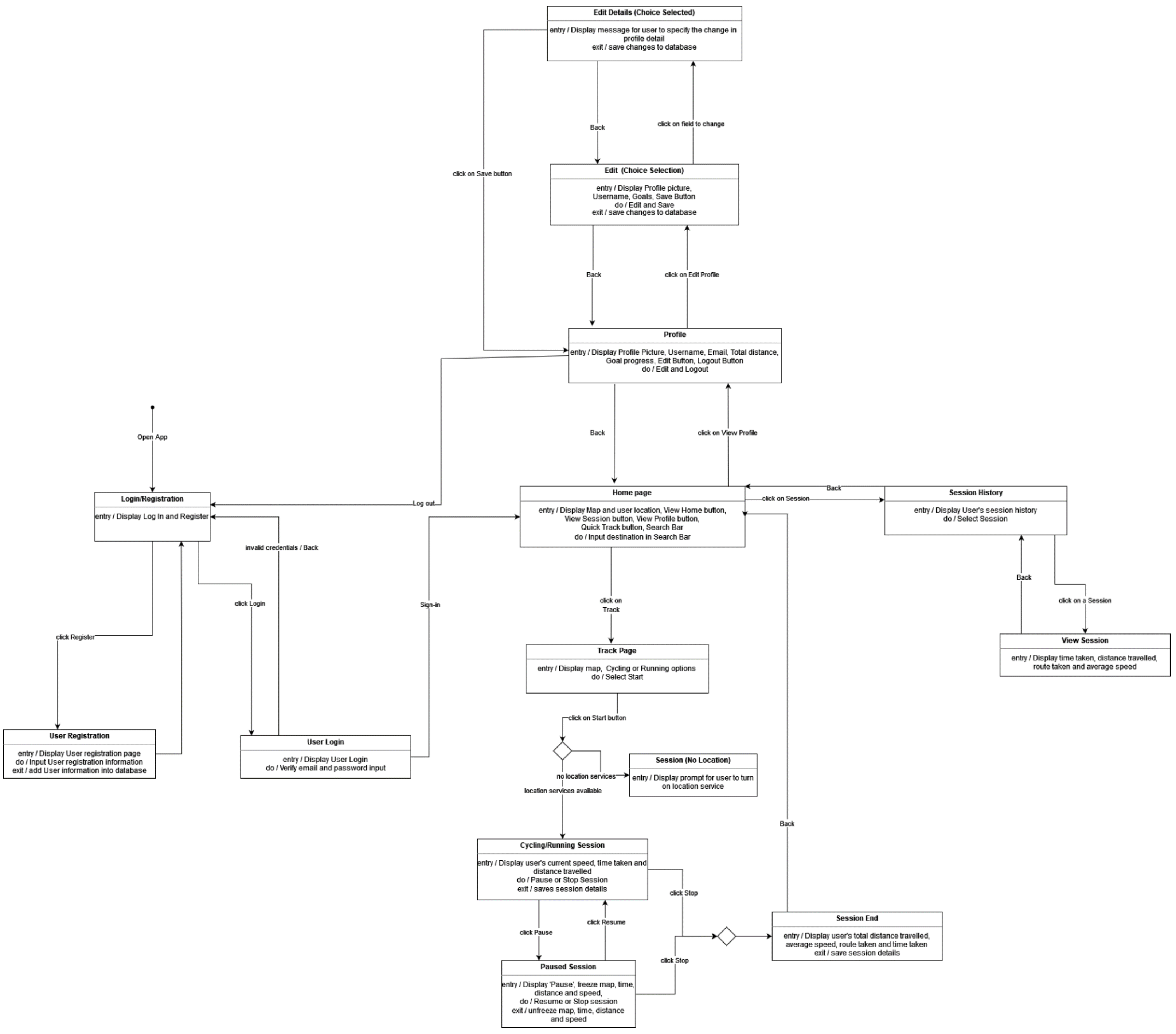
Use Case ID:	SVR.UC003		
Use Case Name:	Verify Login details		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	Server, MongoDB
Description:	The use case begins when the actor receives user details to verify and ends when it has been verified with MongoDB.
Preconditions:	Valid email and associated password entered in the login page.
Postconditions:	Accurately verifies the authenticity of email and password pair with MongoDB and server sends an appropriate message to the system.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The server receives an email and password pair. 2. The server verifies the email is in MongoDB and the password received matches with the password stored in MongoDB. 3. The server sends a “Verified” message to the system. 4. The use case ends.
Alternative Flows:	SVR.UC003.AC1 Email not found 1. The server sends a “Email not found” message to the system. 2. The use case ends. SVR.UC003.AC2 Password mismatch 1. The server sends a “Password mismatch” message to the system. 2. The use case ends.
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

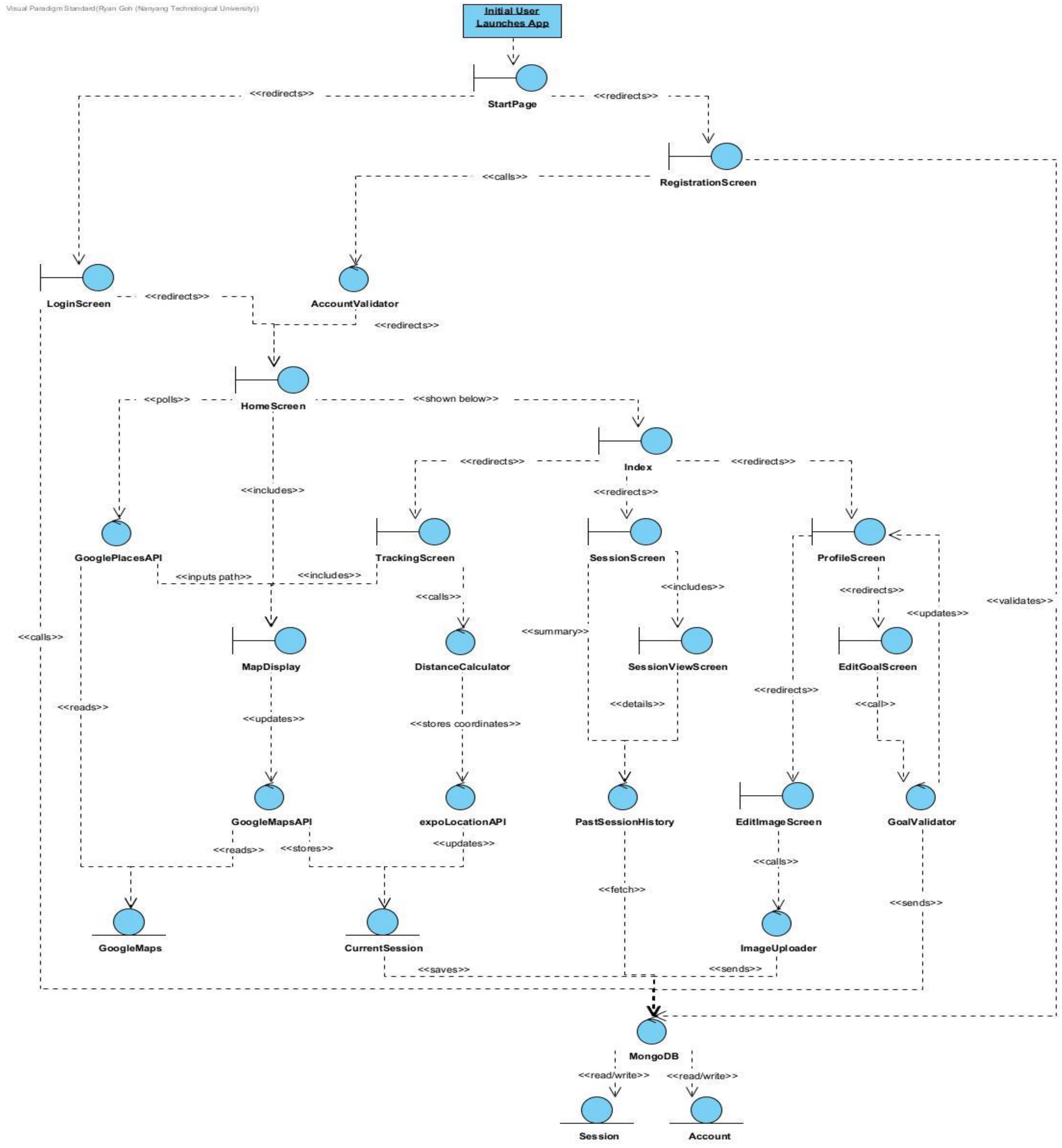
Use Case ID:	SVR.UC004		
Use Case Name:	Save Session details		
Created By:	Arun Ezekiel	Last Updated By:	Kan Huai Feng, Kai
Date Created:	01/02/2023	Date Last Updated:	08/04/2023

Actor:	Server, MongoDB
Description:	The use case begins when the server receives session details - Route, end time, total distance travelled and time taken.
Preconditions:	User is logged in. User has stopped a session on the session page.
Postconditions:	User session details are saved successfully on MongoDB.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The server receives user session details. 2. The server saves the session details, including user id, route, total distance travelled, time taken, and end time on MongoDB. 3. The use case ends.
Alternative Flows:	
Exceptions:	SVR.UC004.EX1 Storage space full 1. The server aborts the case due to not enough space left in storage. 2. The server sends a “Insufficient Storage, please contact an admin” message to the system. 3. The use case ends.
Includes:	
Special Requirements:	
Assumptions:	The user details will always be successfully saved.
Notes and Issues:	

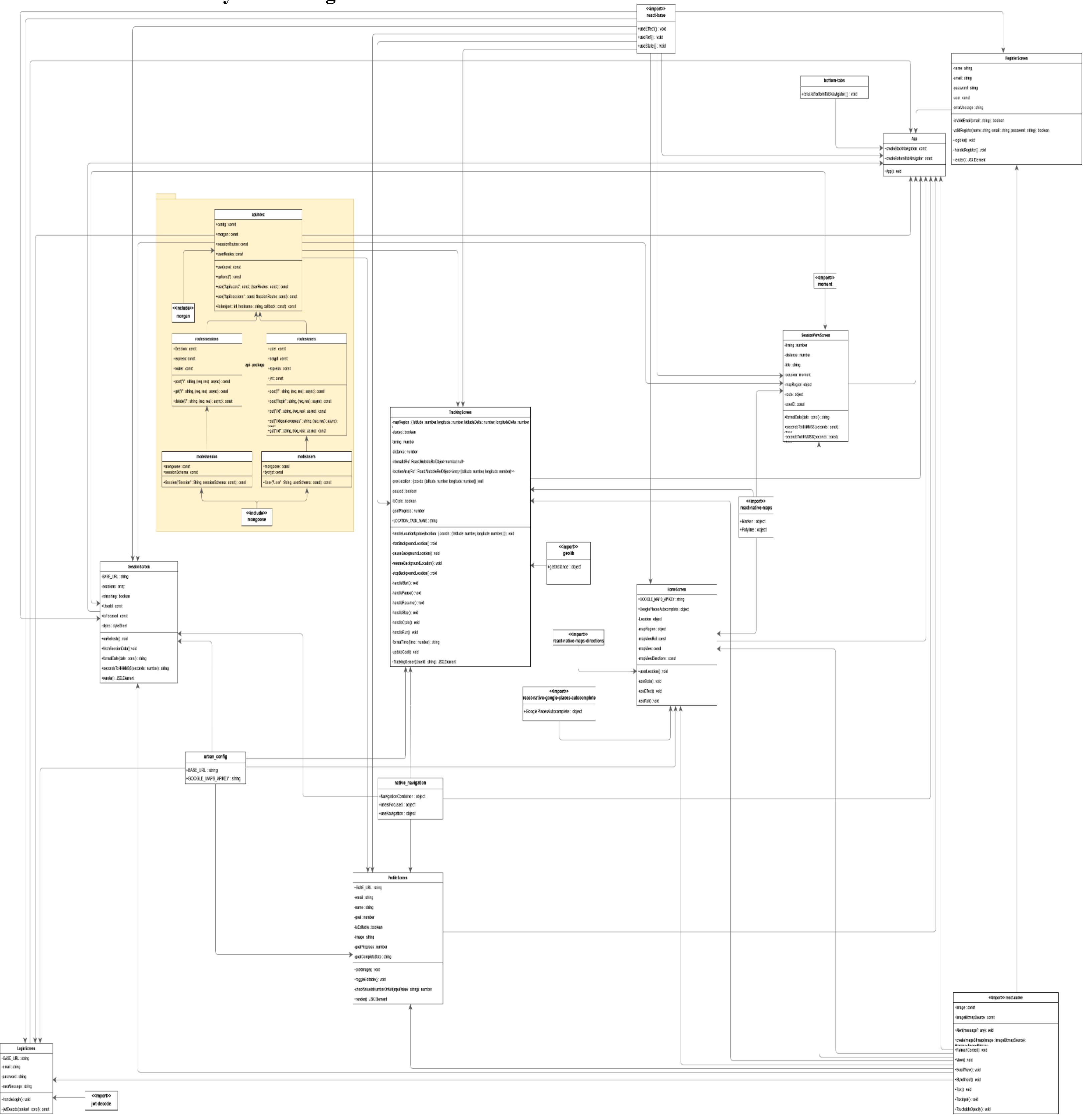
B.3 Dialog Map


Edit Details (Choice Selected)
entry / Display message for user to specify the change in profile detail
exit / save changes to database
Back
click on field to change
click on Save button
Edit (Choice Selection)
entry / Display Profile picture, Username, Goals, Save Button
do / Edit and Save
exit / save changes to database
Back
click on Edit Profile
Profile
entry / Display Profile Picture, Username, Email, Total distance, Goal progress, Edit Button, Logout Button
do / Edit and Logout
Back
click on View Profile
Open App
Login/Registration
entry / Display Log In and Register
Log out
invalid credentials / Back
click Login
click Register
Sign-in
User Registration
entry / Display User registration page
do / Input User registration information
exit / add User information into database
User Login
entry / Display User Login
do / Verify email and password input
Home page
entry / Display Map and user location, View Home button, View Session button, View Profile button, Quick Track button, Search Bar
do / Input destination in Search Bar
Back
click on Session
Session History
entry / Display User's session history
do / Select Session
Back
click on a Session
View Session
entry / Display time taken, distance travelled, route taken and average speed
click on Track
Track Page
entry / Display map, Cycling or Running options
do / Select Start
click on Start button
no location services
Session (No Location)
entry / Display prompt for user to turn on location service
location services available
Cycling/Running Session
entry / Display user's current speed, time taken and distance travelled
do / Pause or Stop Session
exit / saves session details
click Stop
click Pause
click Resume
click Stop
Back
Paused Session
entry / Display 'Pause', freeze map, time, distance and speed,
do / Resume or Stop session
exit / unfreeze map, time, distance and speed
Session End
entry / Display user's total distance travelled, average speed, route taken and time taken
exit / save session details

B.4 Conceptual Class Diagram

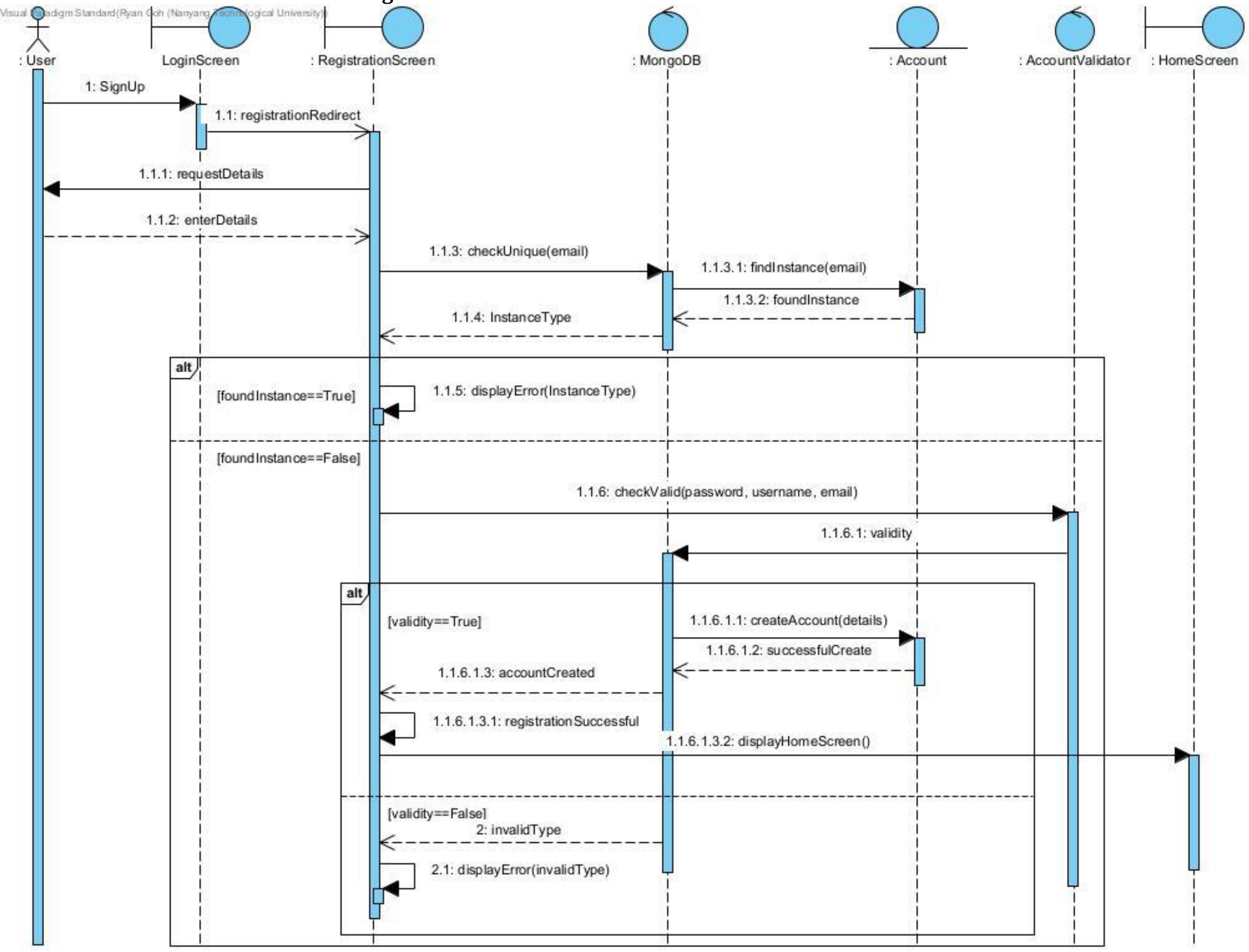


B.5 Entity Class Diagram

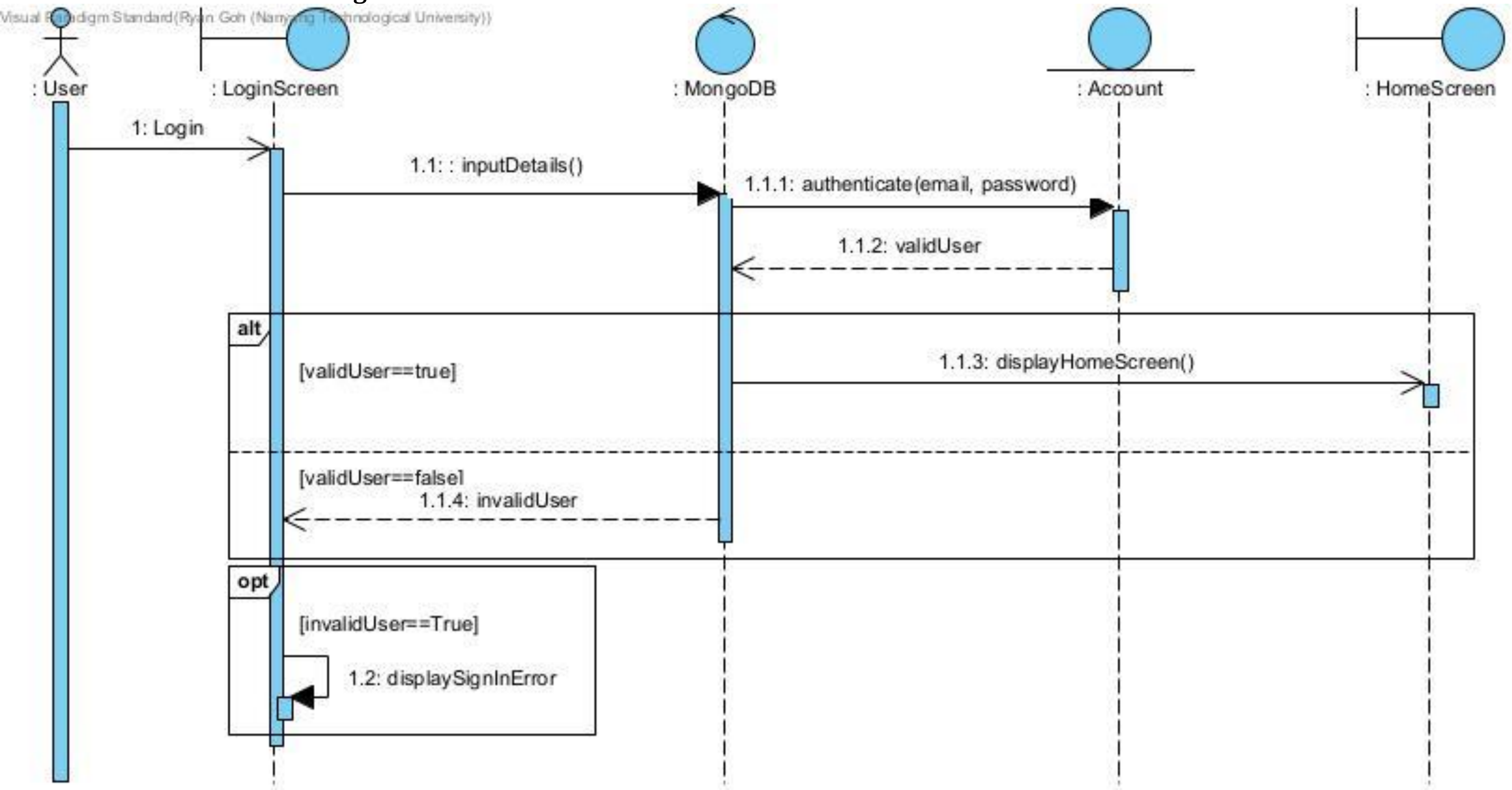

<<import>> react-base
+useEffect() : void
+useRef() : void
+useState() : void
bottom-tabs
+createBottomTabNavigator() : void
App
~createStackNavigation : const
~createBottomTabNavigator : const
~App() : void
RegisterScreen
-name : string
-email : string
-password : string
-user : const
-errorMessage : string
-isValidEmail(email : string) : boolean
-validRegister(name : string, email : string, password : string() : boolean
+register() : void
+handleRegister() : void
+render() : JSX.Element
api/index
+config : const
+morgan : const
+sessionRoutes: const
+userRoutes: const
+use(cors) : const
+options(*) : const
+use("/api/users" : const, UserRoutes : const) : const
+use("/api/sessions" : const, SessionRoutes : const) : const
+listen(port : int, hostname : string, callback : const) : const
<<include>> morgan
routes/sessions
+Session : const
+express: const
+router : const
+post("/" : string, (req, res) : async() : const
+get("/" : string, (req, res) : async() : const
+delete("/" : string, (req, res) : async() : const
api - package
routes/users
+user : const
+bcrypt : const
+express : const
+jwt : const
+post("/" : string, (req, res) : async() : const
+post("/login" : string, (req, res) : async() : const
+put("/:id" : string, (req, res) : async() : const
+put("/:id/goal-progress" : string, (req, res) : async() : const
+get("/:id" : string, (req, res) : async() : const
model/session
+mongoose : const
+sessionSchema : const
+Session("Session" : String, sessionSchema : const) : const
model/users
+mongoose : const
+bcrypt: const
+User("User" : String, userSchema : const) : const
<<include>> mongoose
<<import>> moment
SessionViewScreen
-timing : number
-distance : number
-title : string
-session : moment
-mapRegion : object
-route : object
-userID : const
+formatDate(date : const() : string
+secondsToHHMMSS(seconds : const) : string
-secondsToHHMMSS(seconds : const) : string
TrackingScreen
-mapRegion : {latitude : number, longitude : number, latitudeDelta : number, longitudeDelta : number}
-started : boolean
-timing : number
-distance : number
-intervalIdRef : React.MutableRefObject<number|null>
-locationArrayRef : React.MutableRefObject<Array<{latitude: number, longitude: number}>>
-prevLocation : {coords: {latitude: number, longitude: number}} | null
-paused : boolean
-isCycle : boolean
-goalProgress : number
+LOCATION_TASK_NAME : string
-handleLocationUpdate(location : { coords : { latitude: number, longitude: number } }) : void
-startBackgroundLocation() : void
-pauseBackgroundLocation() : void
-resumeBackgroundLocation() : void
-stopBackgroundLocation() : void
-handleStart() : void
-handlePause() : void
-handleResume() : void
-handleStop() : void
-handleCycle() : void
-handleRun() : void
-formatTime(time : number) : string
-updateGoal() : void
+TrackingScreen(UserId : string) : JSX.Element
<<import>> react-native-maps
+Marker : object
+Polyline : object
<<import>> geolib
+getDistance : object
HomeScreen
+GOOGLE_MAPS_APIKEY : string
+GooglePlacesAutocomplete : object
-Location : object
-mapRegion : object
-mapViewRef: const
-mapView: const
-mapViewDirections : const
+userLocation() : void
-useState() : void
-useEffect() : void
-useRef() : void
<<import>> react-native-maps-directions
<<import>> react-native-google-places-autocomplete
+GooglePlacesAutocomplete : object
SessionScreen
-BASE_URL : string
-sessions : array
-refreshing : boolean
+UserId : const
+isFocused : const
-styles : styleSheet
+onRefresh() : void
+fetchSessionData() : void
+formatDate(date : const() : string
+secondsToHHMMSS(seconds : number() : string
+render() : JSX.Element
urban_config
+BASE_URL : string
+GOOGLE_MAPS_APIKEY : string
native_navigation
+NavigationContainer : object
+useIsFocused : object
+useNavigation : object
ProfileScreen
+BASE_URL : string
-email : string
-name : string
-goal : number
-isEditable : boolean
-image : string
-goalProgress : number
-goalCompleteDate : string
+pickImage() : void
+toggleEditable() : void
-checkValueIsNumberOrNot(inputValue : string) : number
+render() : JSX.Element
<<import>> react-native
+Image : const
+ImageBitmapSource : const
+Alert(message? : any) : void
+createImageBitmap(image : ImageBitmapSource) :
+RefreshControl() : void
+View() : void
+ScrollView() : void
+StyleSheet() : void
+Text() : void
+TextInput() : void
+TouchableOpacity() : void
LoginScreen
-BASE_URL : string
-email : string
-password : string
-errorMessage : string
-handleLogin() : void
-jwtDecode(content : const() : const
<<import>> jwt-decode

B.6 Sequence Diagram

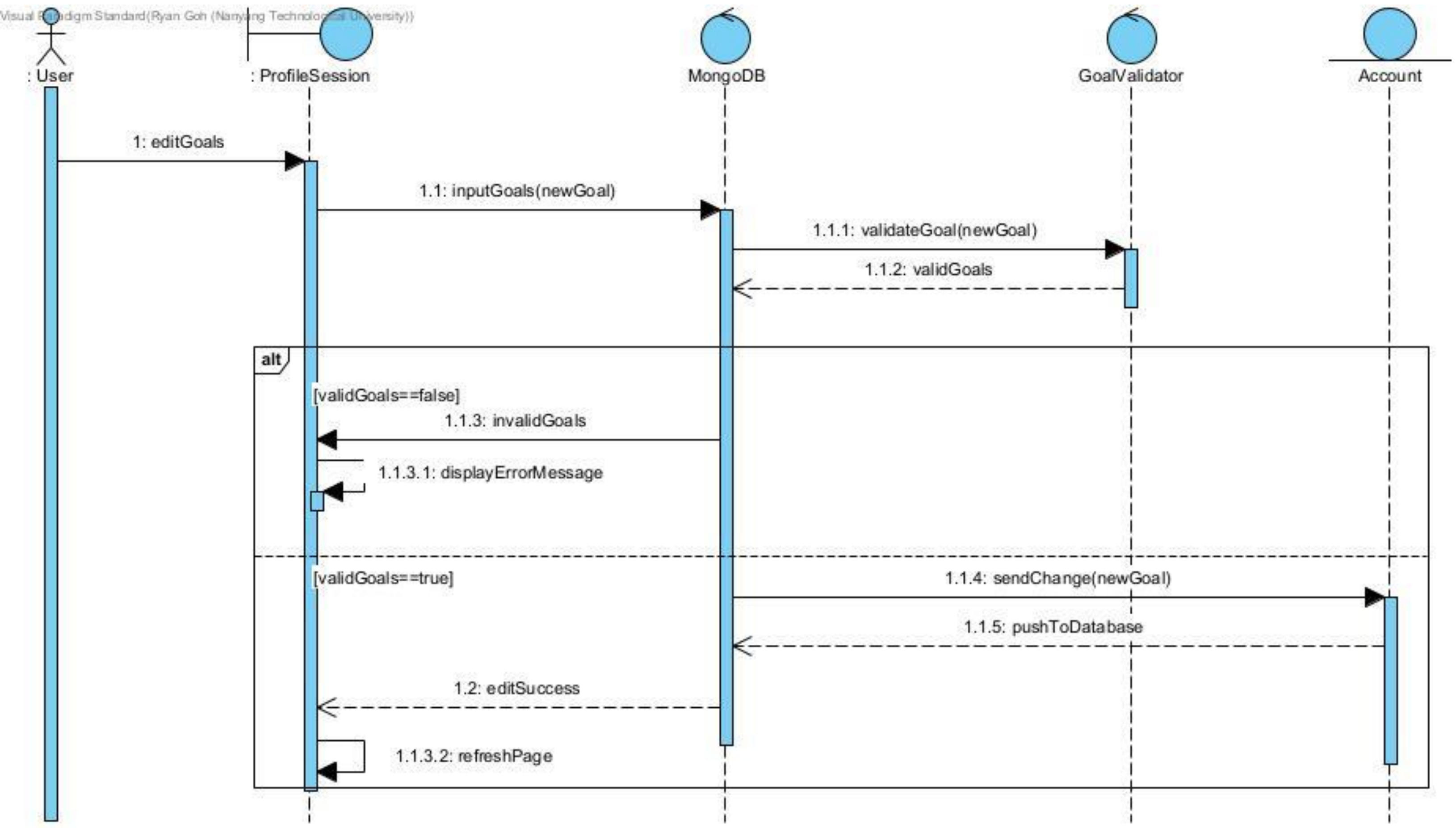
B6.1 Account Registration


Visual Paradigm Standard(Ryan Goh (Nanyang Technological University))
: User
LoginScreen
: RegistrationScreen
: MongoDB
: Account
: AccountValidator
: HomeScreen
1: SignUp
1.1: registrationRedirect
1.1.1: requestDetails
1.1.2: enterDetails
1.1.3: checkUnique(email)
1.1.3.1: findInstance(email)
1.1.3.2: foundInstance
1.1.4: InstanceType
alt
[foundInstance==True]
1.1.5: displayError(InstanceType)
[foundInstance==False]
1.1.6: checkValid(password, username, email)
1.1.6.1: validity
alt
[validity==True]
1.1.6.1.1: createAccount(details)
1.1.6.1.2: successfulCreate
1.1.6.1.3: accountCreated
1.1.6.1.3.1: registrationSuccessful
1.1.6.1.3.2: displayHomeScreen()
[validity==False]
2: invalidType
2.1: displayError(invalidType)

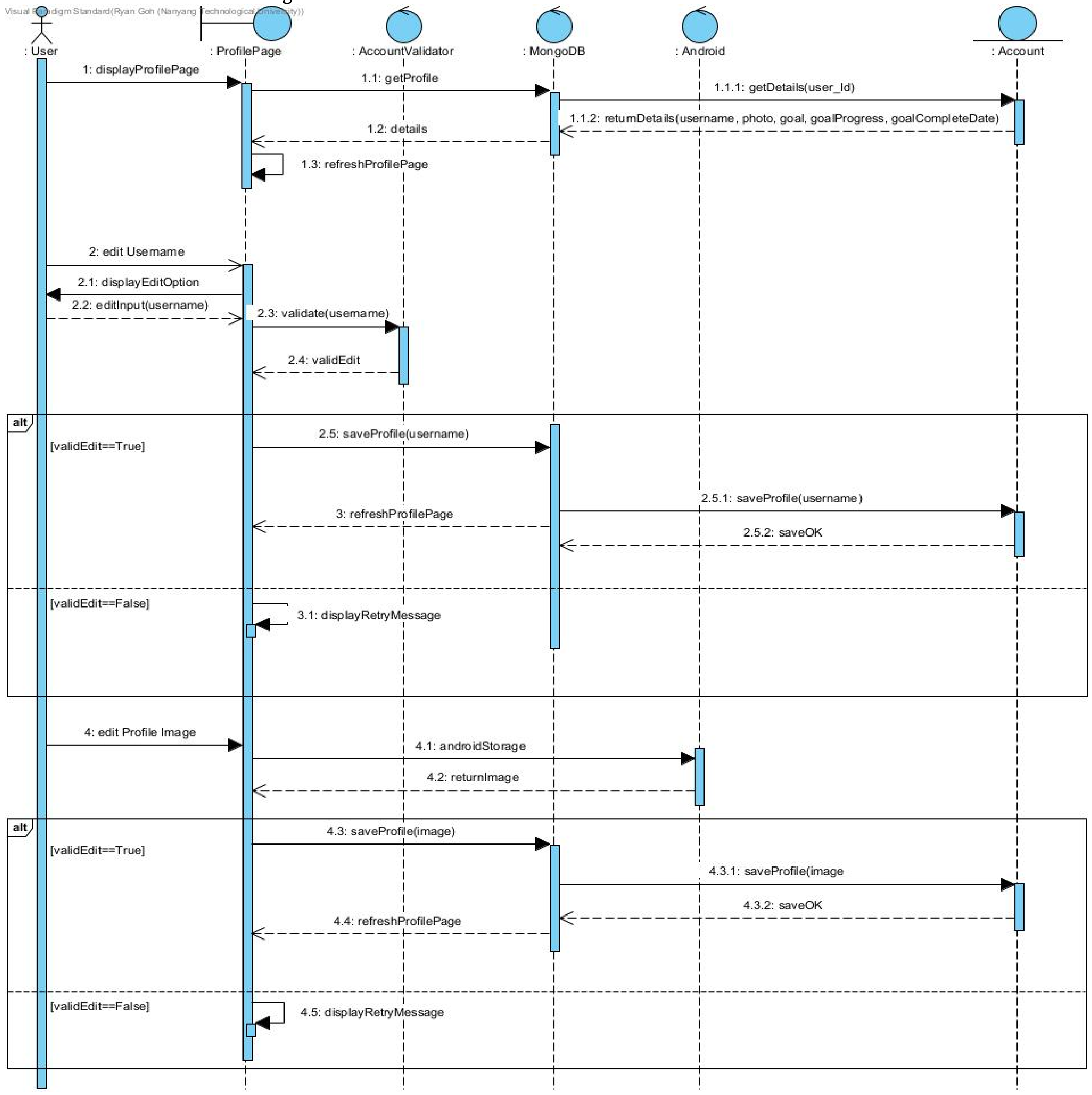
B6.2 Login Account



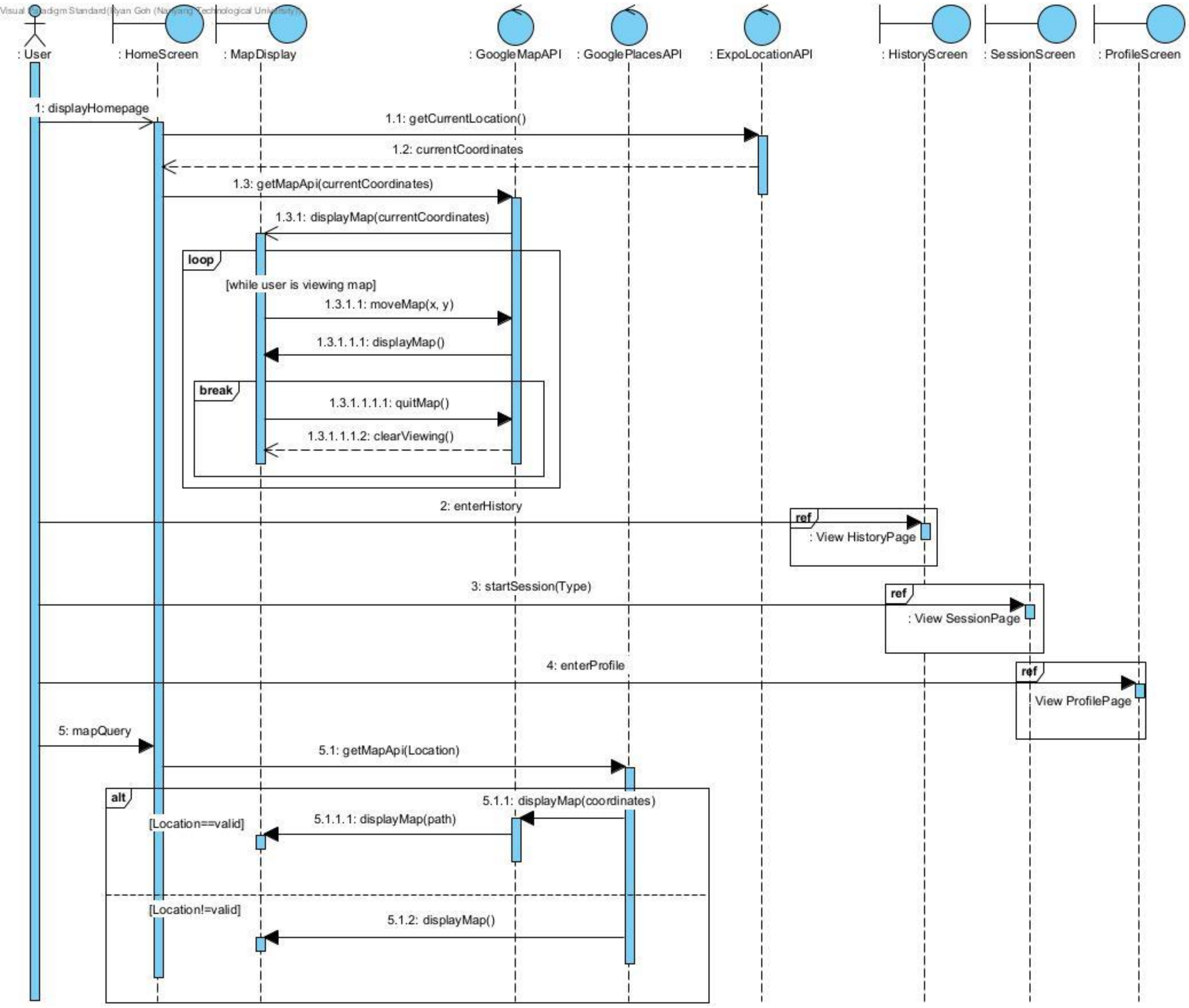
B6.3 Edit Goals



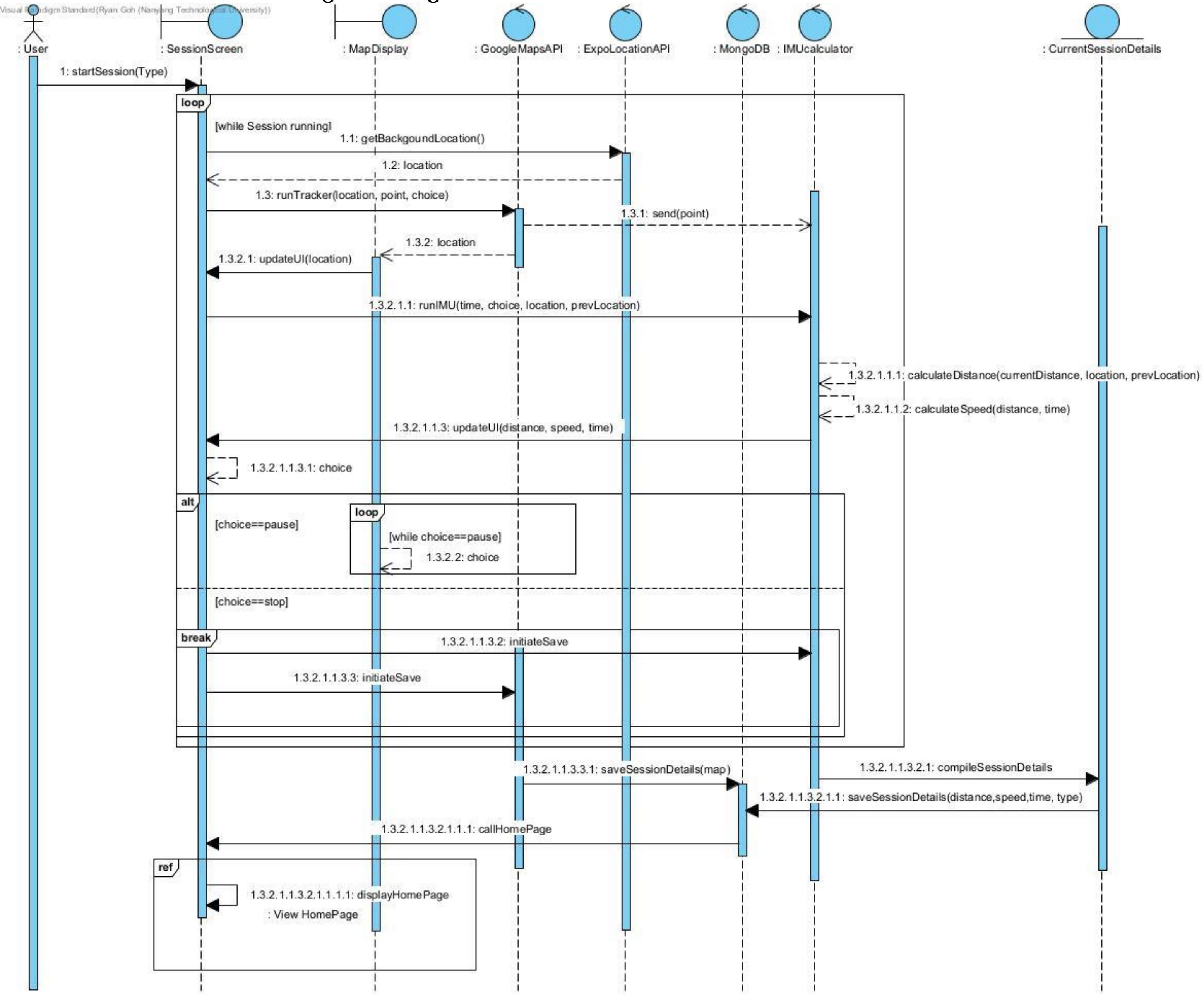
B6.4 Manage Profile



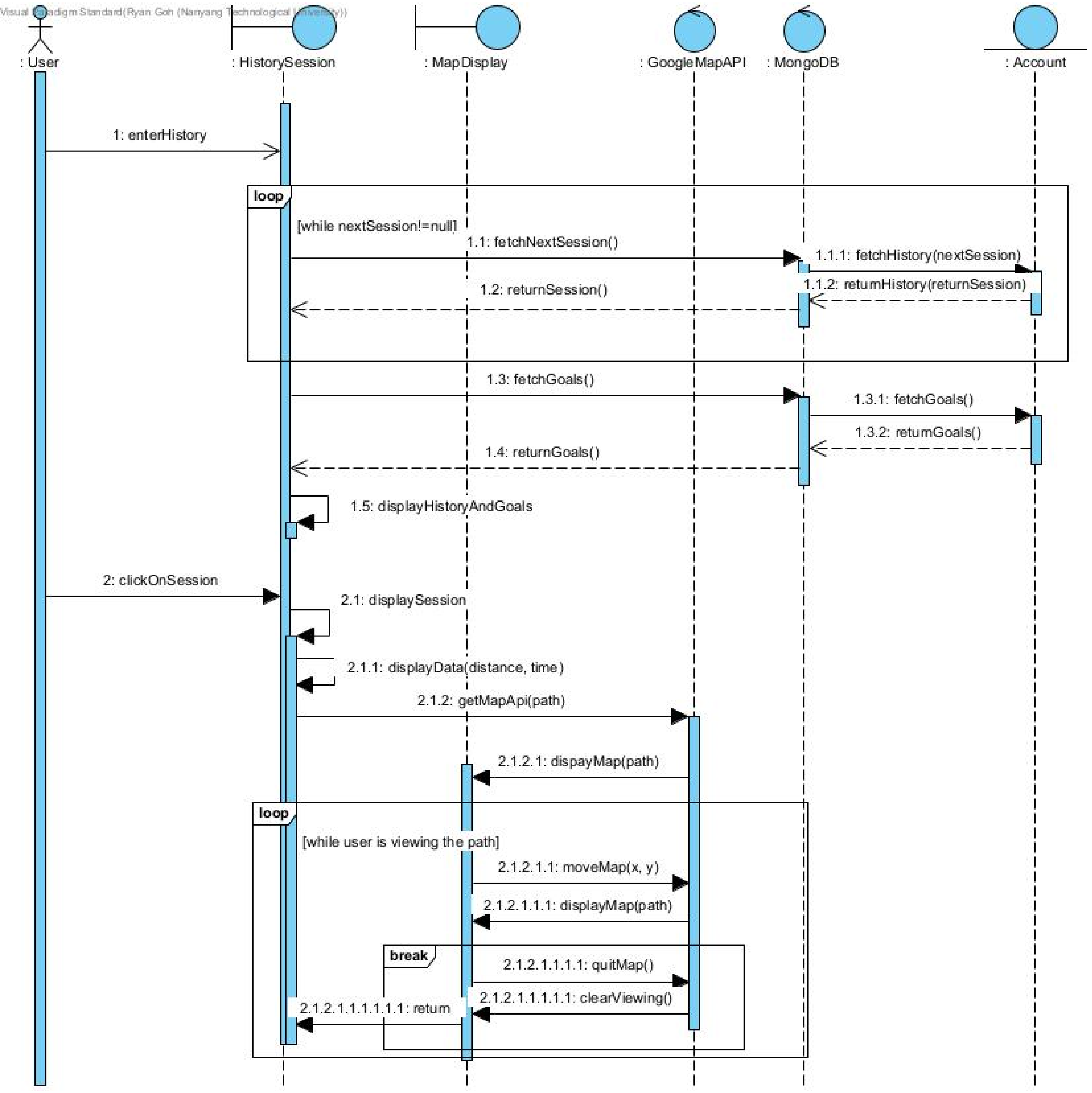
B6.5 Home Screen



B6.6 Manage Tracking


Visual Paradigm Standard(Ryan Goh (Nanyang Technological University))
: User
: SessionScreen
: MapDisplay
: GoogleMapsAPI
: ExpoLocationAPI
: MongoDB
: IMUcalculator
: CurrentSessionDetails
1: startSession(Type)
loop
[while Session running]
1.1: getBackgoundLocation()
1.2: location
1.3: runTracker(location, point, choice)
1.3.1: send(point)
1.3.2: location
1.3.2.1: updateUI(location)
1.3.2.1.1: runIMU(time, choice, location, prevLocation)
1.3.2.1.1.1: calculateDistance(currentDistance, location, prevLocation)
1.3.2.1.1.2: calculateSpeed(distance, time)
1.3.2.1.1.3: updateUI(distance, speed, time)
1.3.2.1.1.3.1: choice
alt
[choice==pause]
loop
[while choice==pause]
1.3.2.2: choice
[choice==stop]
break
1.3.2.1.1.3.2: initiateSave
1.3.2.1.1.3.3: initiateSave
1.3.2.1.1.3.3.1: saveSessionDetails(map)
1.3.2.1.1.3.2.1: compileSessionDetails
1.3.2.1.1.3.2.1.1: saveSessionDetails(distance,speed,time, type)
1.3.2.1.1.3.2.1.1.1: callHomePage
ref
1.3.2.1.1.3.2.1.1.1.1: displayHomePage
: View HomePage

B6.7 View Session



Appendix C: White Box Testing

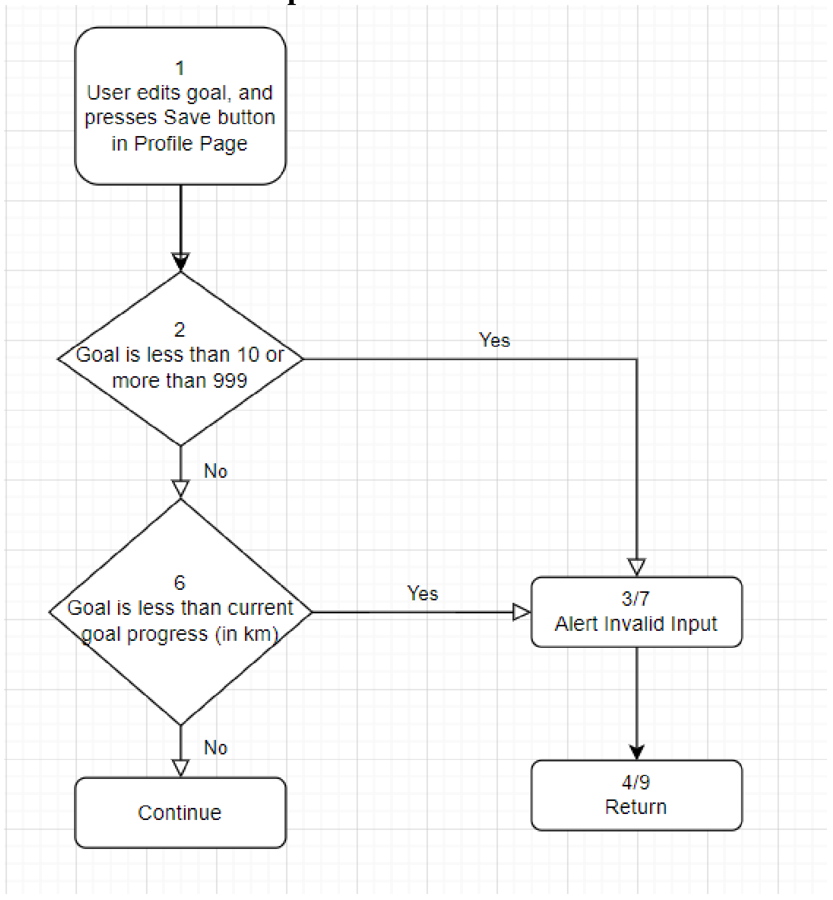
C.1 Edit Goal

Control Flow Test

Code

```
// Check if goal is valid
1   const checkGoal = (goal) => {
2     if (goal < 10 || goal > 999) {
3       Alert.alert("Goal must be between 10-999 km");
4       return;
5     }
6     if (goal < goalProgress / 1000) {
7       Alert.alert("Goal must be between " + goalProgress / 1000 + "-
999  km");
9       return;
10    }
11  };
```

Control Flow Graph


1
User edits goal, and presses Save button in Profile Page
2
Goal is less than 10 or more than 999
Yes
No
6
Goal is less than current goal progress (in km)
Yes
No
3/7
Alert Invalid Input
4/9
Return
Continue

Cyclomatic Complexity

Cyclomatic complexity = decision point + 1 = 2 + 1 = 3

Test cases

I. The user fills in the goal with a valid range and is more than current goal progress.
II. The user fills in a goal outside of the valid range and is more than current goal progress.
III. The user fills in a goal with a valid range but the goal is less than current goal progress.

Basis Path

I. Path 1: 1->2->6-> continue
II. Path 2: 1->2->3->4
III. Path 3: 1->2->6->7->9

Test Results

Path	Result
1	Pass
2	Pass
3	Pass

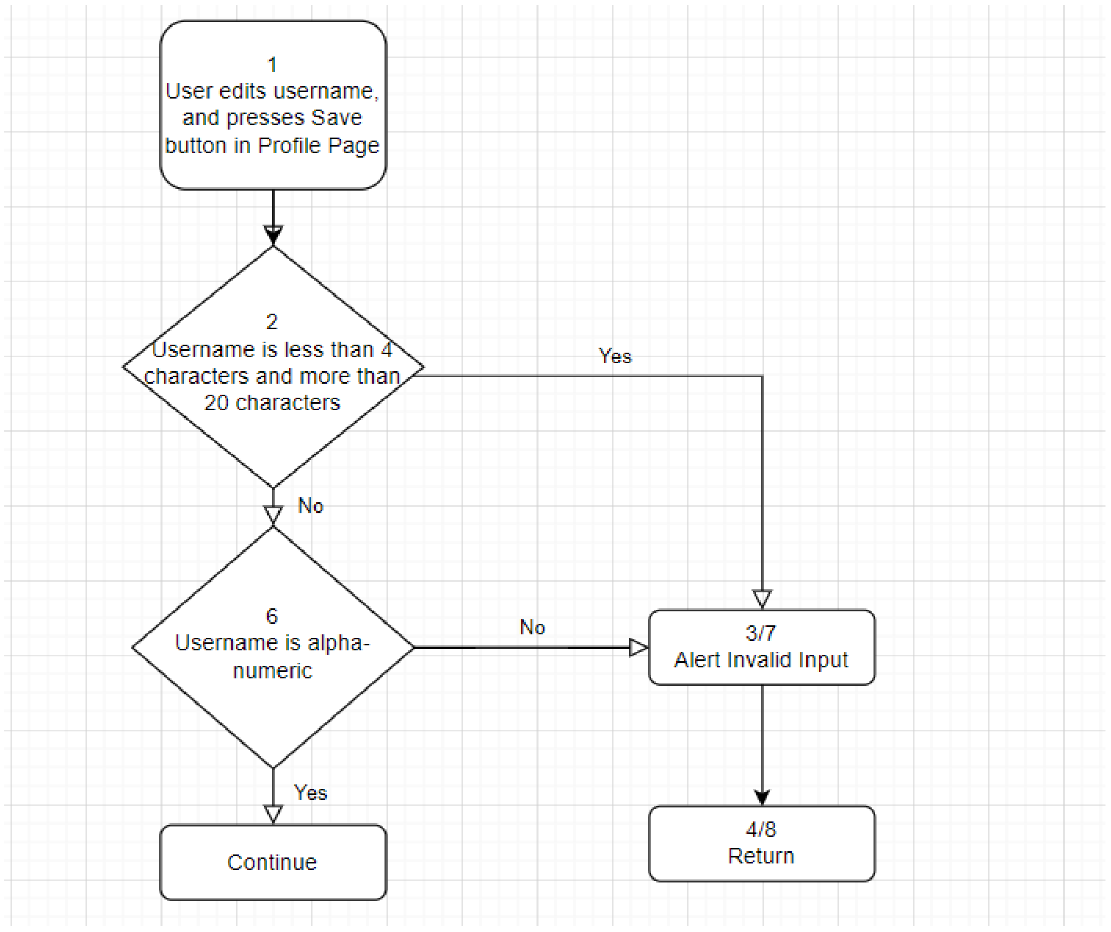
C.2 Edit Username

Control Flow Test

Code

```
//Check if username is valid
1  const checkUsername = (name) => {
2    if (name.length < 4 || name.length > 20) {
3      Alert.alert("Username must be between 4 to 20 characters");
4      return;
5    }
6    if (isAlphanumeric(name) == false) {
7      Alert.alert("Your username cannot contain invalid characters");
8      return;
9    }
10 };
```

Control Flow Graph


1
User edits username, and presses Save button in Profile Page
2
Username is less than 4 characters and more than 20 characters
Yes
No
6
Username is alpha-numeric
No
3/7
Alert Invalid Input
Yes
Continue
4/8
Return

Cyclomatic Complexity

Cyclomatic complexity = decision point + 1 = 2 + 1 = 3

Test cases

I. The user fills in the username within the valid length and is alpha-numeric.
II. The user fills in a username that is outside of the valid length and is alpha-numeric.
III. The user fills in a username that is within the valid length and is not alpha-numeric.

Basis Path

I. Path 1: 1->2->6-> continue
II. Path 2: 1->2->3->4
III. Path 3: 1->2->6->7->8

Test Results

Path	Result
1	Pass
2	Pass
3	Pass

Appendix D: Black Box Testing

D.1 Register Page

Username

	Valid	**Invalid**
Equivalence Class	Username must be between 4-20 characters AND Must be alphanumeric	Blank OR Not between 4-20 characters OR Contain non-alphanumeric characters

Valid Equivalence Class Example

Within Boundary Variables	Peter
Within Boundary Variables	Jack123
Within Boundary Variables	2006420
Outside Boundary Variables	(left empty)
Outside Boundary Variables	Pat
Outside Boundary Variables	Suparmanbinultarmenbinbatman
Outside Boundary Variables	!Peter

Email Address

	Valid	**Invalid**
Equivalence Class	Email with "@", AND "." after "@" included AND Email must not be in database	Blank OR "@" is excluded OR "." is excluded after "@" OR No characters before "@", between "@" and ".", and after "." OR Email in already in database

Valid Equivalence Class Example

Within Boundary Variables	Peter@hotmail.com
Within Boundary Variables	Jack@yahoo.com
Within Boundary Variables	jill@gmail.com
Outside Boundary Variables	(left empty)
Outside Boundary Variables	123456asdasdads
Outside Boundary Variables	Jack@hotmailcom
Outside Boundary Variables	Jackhotmail.com
Outside Boundary Variables	@gg.com

Password

	Valid	**Invalid**
Equivalence Class	Password between 8-20 characters AND	Blank OR

	Must include at least 1 capital letter and 1 number	Not between 8-20 characters OR Does not have at least 1 capital letter or 1 number

Valid Equivalence Class Example

Within Boundary Variables	Peter123
Within Boundary Variables	Peter123@@
Outside Boundary Variables	(left empty)
Outside Boundary Variables	peter
Outside Boundary Variables	123
Outside Boundary Variables	batmanbinsuparmanbinultraman
Outside Boundary Variables	peter123
Outside Boundary Variables	12345678890--12318725387123981237

<u>Test Case (Register Page)</u>

Generic Cases

Case ID	**Input**	**Expected Result**	**Actual Result**
1	Fill up registration page with valid inputs	Alert "Account successfully registered" AND user redirected to Home Screen	Alert "Account successfully registered" AND user redirected to Home Screen
2	Fill up registration page with invalid input(s)	The system alerts the user about the invalid input(s)	The system alerts the user about the invalid input(s)
3	Fill up registration page without filling up all input fields	The system alerts the user to fill up empty input fields	The system alerts the user to fill up empty input fields

Testing

Username	**Email Address**	**Password**	**Expected Result**	**Actual Result**
Peter	<u>peter@gmail.com</u> (Email not found in database)	!Peter123?	Alert "Account successfully registered" AND user redirected to Home Screen	Alert "Account successfully registered" AND user redirected to Home Screen
'empty field'	peter@gmail.com	Peter123	Alert "Username/Email/ Password cannot be Empty!!!"	Alert "Username/Emai l/Password cannot be Empty!!!"
Peter	'empty field'	Peter123	Alert "Username/Email/ Password cannot be Empty!!!"	Alert "Username/Emai l/Password cannot be Empty!!!"
Peter	peter@gmail.com	'empty field'	Alert "Username/Email/	Alert "Username/Emai l/Password

			Password cannot be Empty!!!”	cannot be Empty!!!”
Peter	peter.com (Email found in database)	peter134	Alert “Invalid Email!!!”	Alert “Invalid Email!!!”
Peter	peter123@gmail.com (Email found in database)	!Peter123?	Alert “Email Taken!!!”	Alert “Email Taken!!!”
Peter	peter@gmail.com	peterrrrrr	Alert “At least 1 capital [A-Z] and 1 number [0-9]!!!”	Alert “At least 1 capital [A-Z] and 1 number [0-9]!!!”
Peter	peter@gmail.com	pet@!	Alert “Password must be between 8-20 characters long!!!”	Alert “Password must be between 8-20 characters long!!!”
Peter	peter@gmail.com,	per1234676182u7397123	Alert “Password must be between 8-20 characters long!!!”	Alert “Password must be between 8-20 characters long!!!”
Pet	peter@gmail.com,	per!@#132	Alert “Username must be between 4-20 characters long!!!”	Alert “Username must be between 4-20 characters long!!!”
Petpetpetpetpepetpetpeptpepetpet	peter@gmail.com,	per!@#132	Alert “Username must be between 4-20 characters long!!!”	Alert “Username must be between 4-20 characters long!!!”
Petp!!!etpet	peter@gmail.com	per!@#132	Alert “Your username cannot contain invalid characters!!!”	Alert “Your username cannot contain invalid characters!!!”

D.2 Login Page

Email Address

	Valid	Invalid
Equivalence Class	Email must be in database	Blank or Email is not in the database

Valid Equivalence Class Example

Within Boundary Variables	Peter@hotmail.com (found in database)
Within Boundary Variables	Jack@yahoo.com (found in database)
Within Boundary Variables	jill@gmail.com (found in database)
Outside Boundary Variables	(left empty)
Outside Boundary Variables	123456asdasdads
Outside Boundary Variables	kidz@hotmail.com (not found in database)

Password

	Valid	Invalid
Equivalence Class	Email-Password combination is in the database	Blank OR Email-Password combination is incorrect

Valid Equivalence Class Example

Within Boundary Variables	Peter123 (email found in database, password matches)
Outside Boundary Variables	(left empty)
Outside Boundary Variables	peter (email found in database, password invalid)
Outside Boundary Variables	Kidz123 (email not found in database)

Test Case (Login Page)

Generic Cases

Case ID	Input	Expected Result	Actual Result
1	Fill up login page with valid inputs	Login Successful, user is redirected to main page	Login Successful, user is redirected to main page
2	Fill up login page with invalid input(s)	The system prompts the user to change the invalid input(s)	The system prompts the user to change the invalid input(s)
3	Fill up registration page without filling up all input fields	The system prompts the user to fill up empty input fields	The system prompts the user to fill up empty input fields

Testing

Email	**Password**	**Expected Result**	**Actual Result**
peter@gmail.com	Peter123	Login Successful, user is redirected to main page	Login Successful, user is redirected to main page
‘empty field’	Peter123	Alert “Email/Password cannot be Empty!!!”	Alert “Email/Password cannot be Empty!!!”
peter@gmail.com	‘empty field’	Alert “Email/Password cannot be Empty!!!”	Alert “Email/Password cannot be Empty!!!”
peter.com (not found in database)	peter134	Alert ”Invalid Email!!!”	Alert ”Invalid Email!!!”
kidz@hotmail.com (not found in database)	peter123	Alert ”Invalid Email!!!”	Alert ”Invalid Email!!!”
peter@gmail.com (found in database)	Peter!234 (does not match Email-Password combination)	Alert ”Invalid Password!!!”	Alert ”Invalid Password!!!”