Software Requirements Specification

for

<ParkWhere>

Version 1.0 approved

Prepared by <Aaron Lim Wee Him>

<Nanyang Technological University, Team 2>

<13/4/2024>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Aaron Lim	13/4/2024	Final Submission	1.0

1. Introduction

1.1 Purpose

This Software Requirement Specification (SRS) document is intended for the **ParkWhere** web application, build version 1.0. The purpose of this SRS document is to describe the requirements specifications for the **ParkWhere** web application to facilitate the development and production process for all stakeholders. This document will include the system features, limitations, functional and non-functional requirements, as well as use case descriptions. The **ParkWhere** web application aims to help users to find the nearest carparks either from current location or searched location and allow users to make informed choices from there. This **ParkWhere** web application is built using the following frameworks, front-end framework being React JS, back-end framework being Node JS as well as the usage of Firebase as a database system.

1.2 Document Conventions

This section describes all standards and typographical conventions that we will be following when writing the SRS document.

Refer to appendix A for the list of definitions (Data dictionary) for special terms used during the description of our report.

This document follows the IEEE standards. Priorities of higher-level requirements are inherited by detailed-level requirements.

Font	Arial, Black font colour
Heading	Times, Bold, Size 18
Subheading	Times, Bold, Size 14
Text	Size 11

1.3 Intended Audience and Reading Suggestions

This section will describe the different types of readers that the document is intended for and will provide suggestions to the sequence for reading the document.

This SRS document is meant for all stakeholders such as **ParkWhere** users, **ParkWhere** development team, **ParkWhere** testing team as well as **ParkWhere** project manager.

This document begins with the introduction where it will state the purpose and product scope about the project. Thereafter, it will be followed by the overall description of the product which includes the application functionalities, several design constraints and assumptions of application. It will then be followed by the introduction of both system functionalities and non-functional requirements of the product, ending with an appendix that encompasses the glossary and analysis models.

All stakeholders should begin reading the document from the introduction section - 1.1 Purpose, 1.2 Document Conventions as well as Appendix A (Data dictionary) to get a glimpse of what the product is about and have a brief introduction to the product.

The **ParkWhere** development team can then proceed with reading section 2. Section 2 will provide the development team with a high-level description of product perspective, functionalities that will be useful in helping the development team to build their product. After which, section 4 (System Feature) in which developers can have a better understanding about the functionalities and features of the application.

On the other hand, the **ParkWhere** users, testing team and project managers can read through this document in sequential order.

1.4 Product Scope

This section provides a short description of the product's purpose, benefits, objectives and goals.

Due to the scarcity of parking spots, particularly in densely populated urban areas, finding a spot in Singapore can be a difficult chore. In Singapore, owning a car is not only costly but also a burden when it comes to parking and maintenance, which deters many residents from considering it.

Our team is creating the **ParkWhere** web application, a CarPark Finder, in response to the growing difficulty in locating parking spots in Singapore. This platform's real-time data, which is derived from public databases, is intended to make the process of finding open parking spaces more efficient.

With up-to-date information on parking availability, costs, and locations throughout the city, **ParkWhere** will enable users. With the use of their present location or intended destination as input, users may quickly retrieve a list of parking possibilities that are close by, along with important details like cost, hours of operation, and any restrictions.

Our target audience for this application comprises road users. Whether they require parking for a short duration or an extended period, **ParkWhere** will facilitate the discovery of the most convenient and available parking spaces.

ParkWhere will provide more tools to improve user experience in addition to streamlining the parking space search. These consist of the opportunity to reserve parking in advance for extra convenience, real-time information on parking availability, and aid with navigation to the designated parking spot. Our goal is to enhance Singaporeans' mobility and reduce the stress that comes with locating parking in the city by offering a holistic solution to the parking conundrum.

1.5 References

- IEEE 830-1998 Template
- React JS Documentation: https://reactjs.org/docs/getting-started.html
- Node JS Documentation: https://nodejs.org/en/docs
- Firebase Documentation: https://firebase.google.com/docs/
- Google Maps API: https://developers.google.com/maps/documentation/javascript
- Carpark API: https://beta.data.gov.sg/

2. Overall Description

2.1 Product Perspective

ParkWhere web application is a new, standalone, self-contained web application. It is a new web application that aims to help users to find the best car park to use near their destination. The application uses the Google Maps API to allow users to input their destination to search for car parks nearby and navigate to any of these car parks. The application can also display information of these car parks such as the live lot availability and hourly rates.

The application is inspired by existing applications that can be used to find car parks such as Waze, but aims to fill the gaps that these applications have, such as the lack of car park information.

2.2 Product Functions

This section will summarise the major functions that the product must perform or must let the user perform.

The **ParkWhere** web application product functionalities will be broken down into three main sub-categories - Accounts, Search, Favourites.

A. Accounts
- Users can register for an account
- Users can log in to their registered account
- Users can log in using their Google account
- Users can recover their password in the case if users forgot their password
- Users can log out of their account

B. Search
- Users can input a location to search for car parks nearby

C. Favourites
- Users can add a car park to their favourites list
- Users can rename car parks in their favourites list
- Users can remove car parks from their favourites list

D. Profile
- Users can change their password in their Profile page
- Users can edit their name in their Profile page

2.3 User Classes and Characteristics

This section will describe the user classes and characteristics of the applications that are based on frequency of use, subset of product function used, technical expertise, security or privilege levels. The main users for our application are people with driving licences and also families who are on the lookout for car parking spaces.

A. People who own a car

Attributes	**Description**
Frequency of use	High
Subset of product functions used	All
Technical expertise	Low
Characteristics	These are people who own a private vehicle. Hence, **ParkWhere** will come into good use where it allows them to find their most preferred car parks when they travel.

B. People who rent a car

Attributes	**Description**
Frequency of use	High
Subset of product functions used	All
Technical expertise	Low
Characteristics	These are people who rent a vehicle. Hence, **ParkWhere** will come into good use where it allows them to find their most preferred car parks when they travel.

2.4 Operating Environment

This section will provide the description of the environment in which the software will operate, including the hardware platform, operating system and versions together with other software components that must peacefully coexist.

Product Environment of **ParkWhere**: The web application will need to be run using common web browsers like Google Chrome or FireFox.

Development Environment of **ParkWhere**:

Development Environment	Description
Front-end: React.js	React.js framework is an open-source JavaScript framework and library developed by Facebook. It is used for building interactive user interfaces and web applications efficiently. React.js allows us to develop our application by creating reusable components. Hence, React.js is suitable to be used to design and build web applications.

	ReactJS (version 18.2.0)
Back-end: Node JS	Node.js is an open source server environment which is found on multiple platforms such as Windows, Linux, Mac OS etc. It is an asynchronous event-driven JavaScript runtime which is designed to build scalable network applications. NodeJS (version 18.15.0)
Database: Firebase	Firebase Realtime Database is a cloud-based NoSQL database by Google Firebase Realtime Database provides powerful real-time synchronization, allowing multiple clients to listen for changes to specific data nodes. When changes occur, the database automatically pushes updates to all connected clients, ensuring that everyone has the latest information.

2.5 Design and Implementation Constraints

- Limitations

Limitations	Details
Google Maps API	Time taken to retrieve location or data from Google Maps API might take longer than expected due to the use of free service.

- Design Standards

Design Standards	Details
Programming standards	● Each frontend component needs to have its own folder that consists of .js file and .css file for better management of different components. ● Avoid deep nesting in codes. This will make it easier to follow and read as well as to debug. ● Use proper indentation to improve readability of code. ● Leave comments to describe code function so that it can guide other developers when working on the code. ● All non-class variables must adopt the camelCase naming conventions. ● All class variables must adopt the PascalCase naming conventions.

User Interface Standards	● All User Interface design must adopt the same colour and theme scheme throughout all the pages. ● All User Interface design needs to adhere to the pre-approved or considered user interface design.

2.6 User Documentation

A demo video on the flow of the application will be attached and shown to provide users with explanations on the navigation of the application.

To further ease developers in setting up the web application, README files with specific instructions to set up the environment for both front-end and back-end would be provided for the users.

2.7 Assumptions and Dependencies

This section will further discuss any assumed factors that could affect the requirements stated in the SRS.

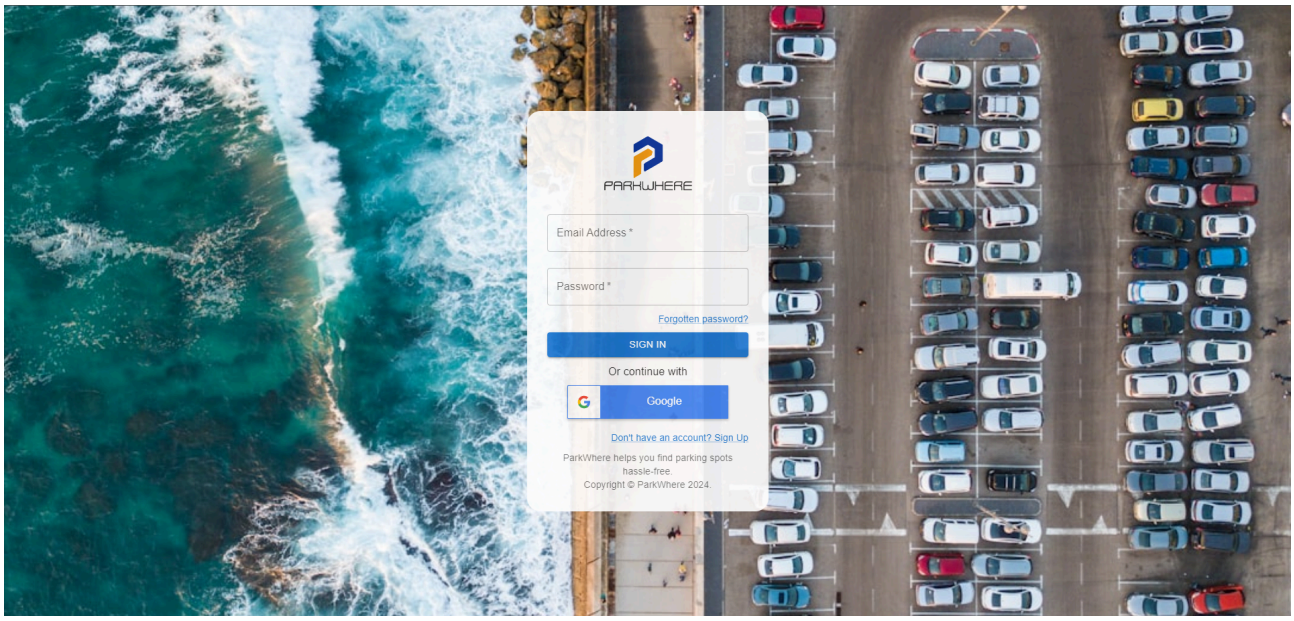
- We assume that all data that is extracted from the APIs are accurate and up-to-date.
- The application assumes that users are connected to a strong internet connection.
- The application assumes that users will use it for its intended purpose of finding car parks

3. External Interface Requirements

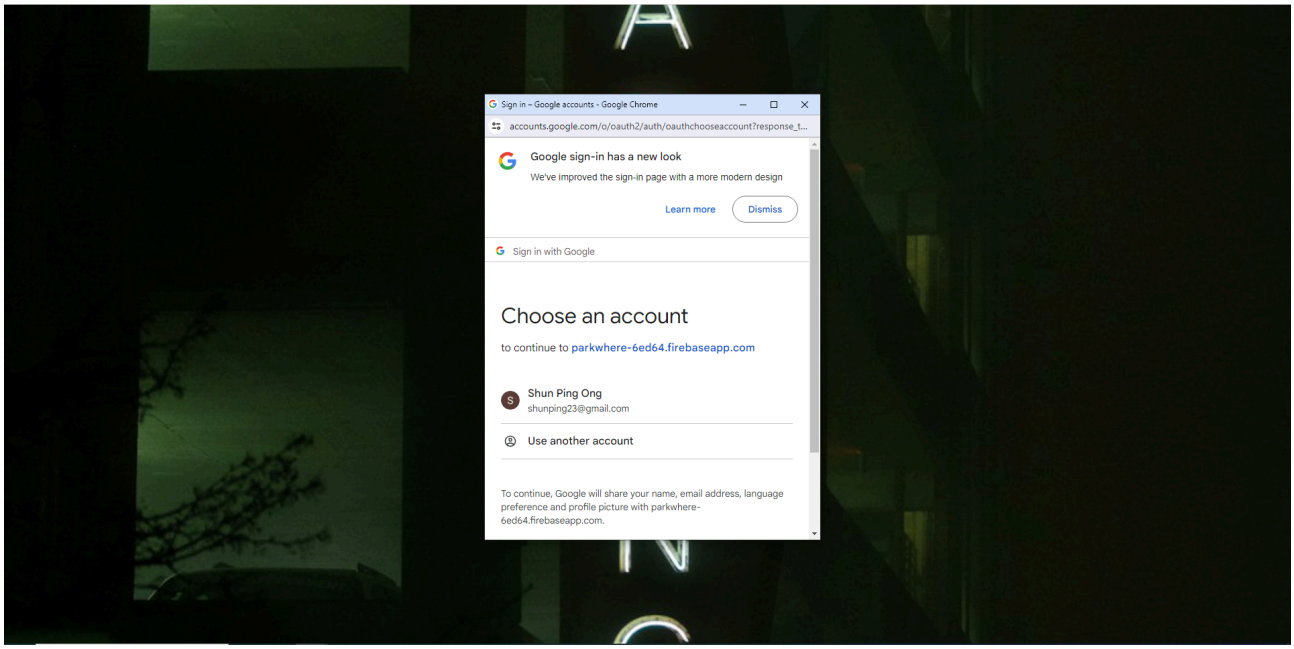
3.1 User Interfaces

This segment will provide a description of the logical characteristic of each interface. Sample screen images and user interfaces will be shown for better illustration.

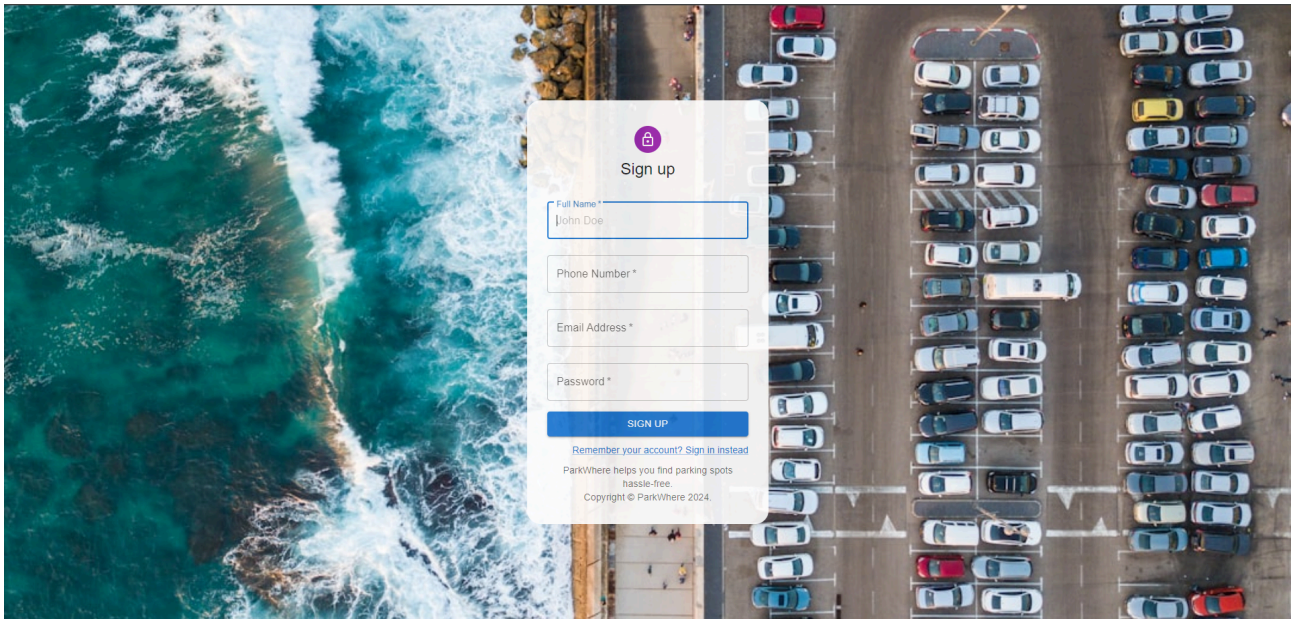
A. Login Page



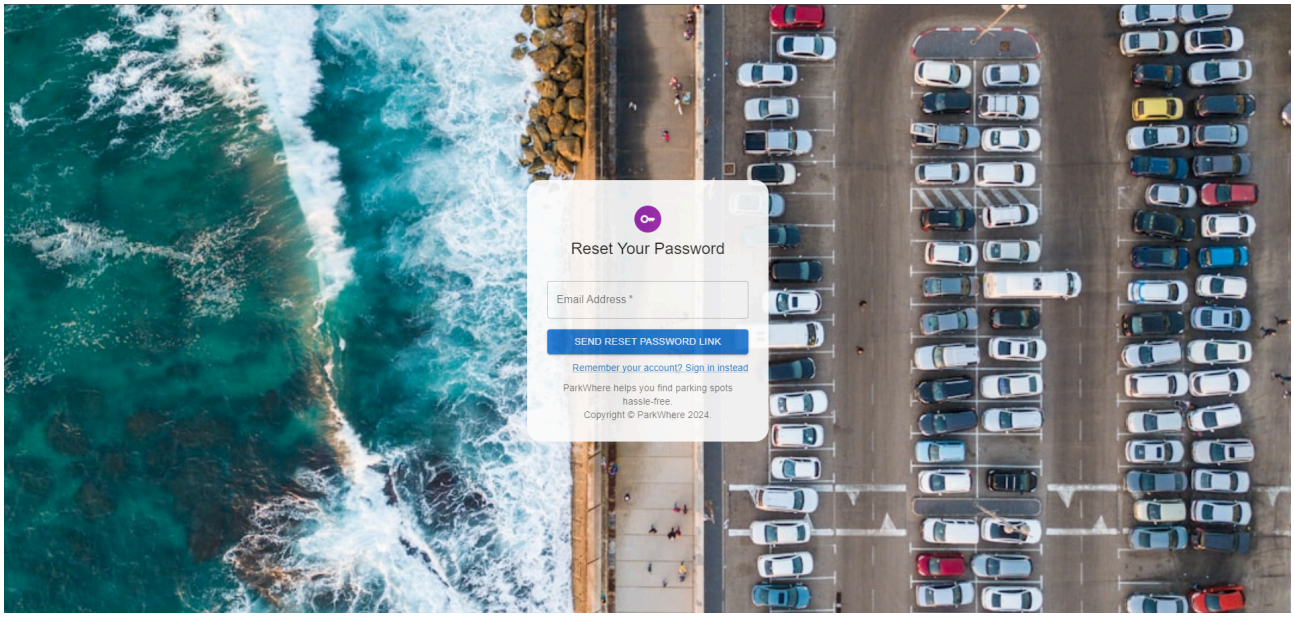
B. Google Login Page



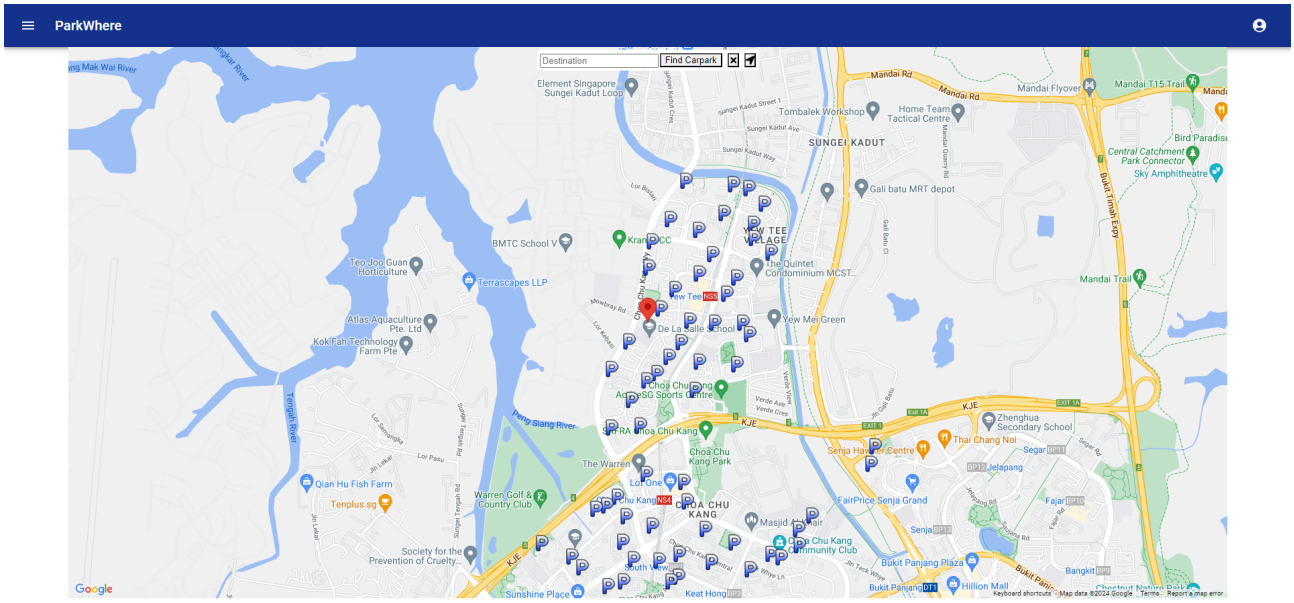
C. Register Account



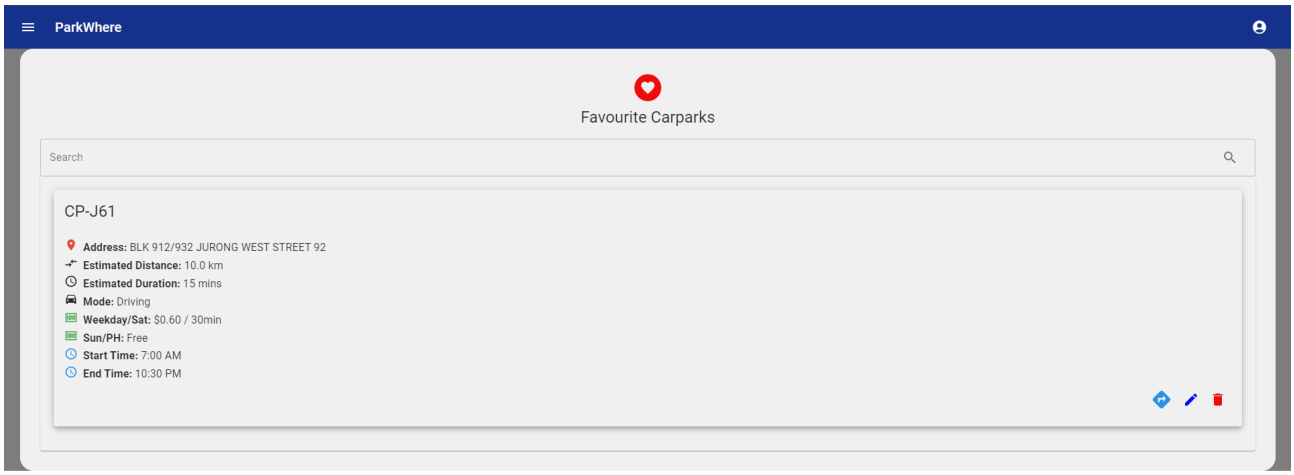
D. Forgot password page



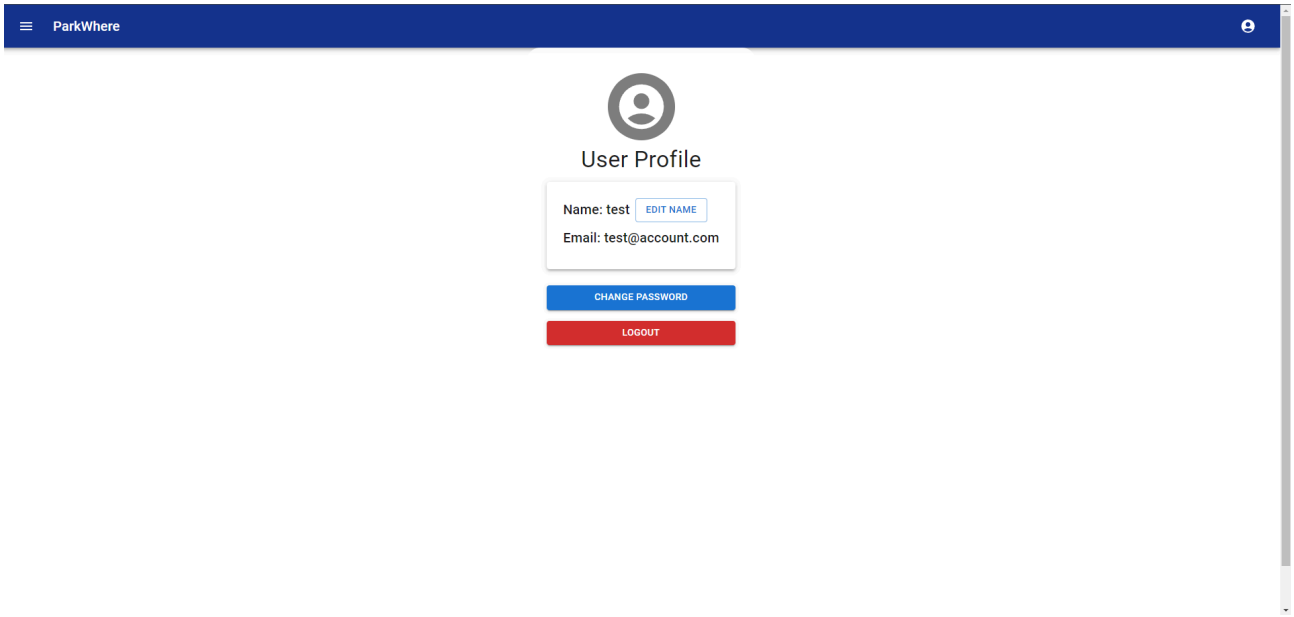
E. Home Page after user logs in



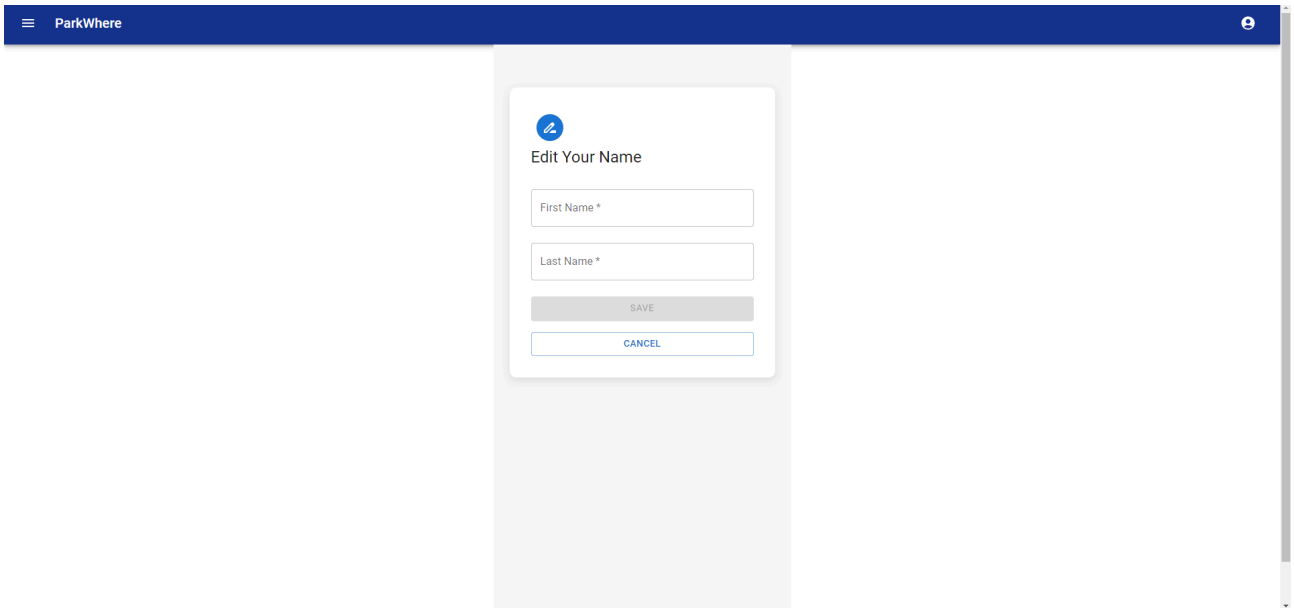
F. Favourites Page



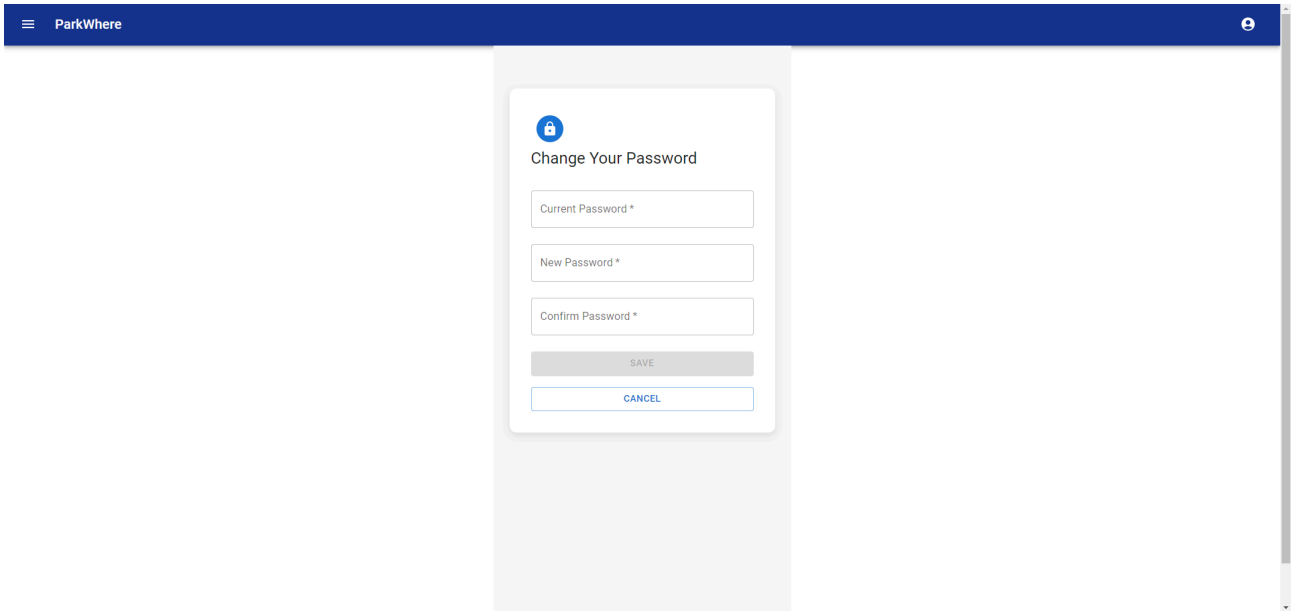
G. Profile Page



H. Edit Name Page



I. Change Password Page



3.2 Hardware Interfaces

This section will cover all hardware interface requirements for the **ParkWhere** application to achieve its desired functionalities. This includes the supported device types that are needed, the nature of data and control interactions between the software and hardware as well as communication protocols being used.

A. Client-side Requirements

The **ParkWhere** web application will support all desktop computers or laptops as well as being compatible with web services on smartphones. This device must support the usage of a desktop browser such as GoogleChrome or Firefox.

B. Server-side Requirements

The **ParkWhere** backend server will be hosted and run on a server-computer, utilizing Firebase as the backend database solution. The backend server will handle fundamental CRUD (Create, Read, Update, Delete) operations and establish connections with the client service. The Firebase Realtime Database will provide a cloud-hosted solution for storing and managing data. This setup ensures seamless integration and scalability, allowing for efficient management of ParkWhere's data.

3.3 Software Interfaces

This segment will describe the connections between this product and other software components which comprises databases, operating systems, tools, libraries or integrated commercial components.

A. Software Components

The **ParkWhere** web application is using FireBase as a back-end database to handle all the querying of data in the web application. The back-end server is implemented using the node.js framework which will help to perform functionalities, the front-end that is implemented using React JS.

B. Software Architecture

The **ParkWhere** software architecture is following the 3-Layered Architecture design pattern.

3.4 Communications Interfaces

This segment will discuss all the associated communication interfaces for this web application to be working.

The **ParkWhere** web application is following the REpresentational State Transfer which is an architectural style for allowing easier communication between systems. This allows the implementation of client and server to be done independently. In the REST architecture, clients will send requests to retrieve or modify data, and servers send responses to these requests.

Communication from client to server will invoke the GET and POST requests. The **ParkWhere** web application will require communicating with an existing email account to post email during forget password test cases.

4. System Features

This section will include all system feature description and use cases.

4.1 Register

Use Case ID:	01		
Use Case Name:	Register		
Created By:	Yi Heng	Last Updated By:	Aaron
Date Created:	30/01/2024	Date Last Updated:	03/02/2024

Actor:	User, System, Google API
Description:	Register for a new account (locally and Google)
Preconditions:	User had downloaded and opened the application
Postconditions:	1) User account created 2) App displays login page
Priority:	High
Frequency of Use:	Low
Flow of Events:	Local: 1) User enters login page 2) User clicks on 'Create Account' 3) User inputs email as user ID 4) User inputs password 5) App verifies user details 6) App saves user details in database 7) App displays successful account creation Google: a) User enters Google login page b) User clicks on 'Create Account' c) User clicks on 'Link Account' d) User inputs Google email address e) User inputs password f) Google API verifies user details g) App displays successful account creation
Alternative Flows:	01-AF-S3 User inputs invalid email 1) App displays 'Error, please input valid email' 2) User returns to step 3) 01-AF-S3 User inputs email that already exists 1) App displays 'Error, email already exists' 2) User returns to step 3) 01-AF-S4 User inputs invalid password 1) App displays 'Error, password does not meet requirements' 2) User returns to step 3) 01-AF-Sd User inputs invalid Google email a) App displays 'Error, please input valid email'

	b) User returns to step d) 01-AF-Sd User inputs email that already exists a) App displays 'Error, email already exists' b) User returns to step d) 01-AF-Se User inputs invalid password a) App displays 'Error, password does not meet requirements' b) User returns to step e)
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

1. Users must be able to register for an account on the system manually or using their Google Account.
    1.1. The system must display text fields for the user to enter his/her information on account creation for manual creation.
        1.1.1. The text fields must consist of Full name.
        1.1.2. The text fields must consist of Email.
        1.1.3. The text fields must consist of Password.
        1.1.4. The text fields must consist of Confirm password.
        1.1.5. The user must fill in all the text fields before clicking "Create your account."
            1.1.5.1. The system must verify the information when the user clicks the button.
                1.1.5.1.1. The email given has never been registered in the system.
                1.1.5.1.2. The email given must be in the correct format.
                1.1.5.1.3. The password given must contain at least 1 uppercase character.
                1.1.5.1.4. The password given must contain at least 1 lowercase character.
                1.1.5.1.5. The password given must contain at least 8 characters.
                1.1.5.1.6. The password given must contain at least 1 number.
                1.1.5.1.7. The system must provide error messages describing the reason the account creation is rejected.
    1.2. The system must display text fields for the user to enter his/her information on account creation for Google creation.
        1.2.1. The text fields must consist of Google Email.
        1.2.2. The text fields must consist of Google Password.
        1.2.3. The user must fill in all the text fields before clicking "Sign Up."
            1.2.3.1. The Google Firebase system must verify the information when the user clicks the button.
    1.3. The system must create an account for the user upon verification.
    1.4. The system must log the user into the main page of the system.

4.2 Login

Use Case ID:	02		
Use Case Name:	Login		
Created By:	Yi Heng	Last Updated By:	Aaron
Date Created:	30/01/2024	Date Last Updated:	03/02/2024

Actor:	User, System
Description:	User must log in to the system
Preconditions:	User must already be registered in the application
Postconditions:	1) User logged in 2) App displays main menu
Priority:	High
Frequency of Use:	Medium
Flow of Events:	Local: 1) User enters login page 2) User inputs email as user id 3) User inputs password 4) User clicks on 'Login' 5) App verifies user details 6) App goes to main menu Google: a) User enters Google login page b) User inputs Google email as user id c) User inputs password d) User clicks on 'Login' e) App verifies user details f) App goes to main menu
Alternative Flows:	02-AF-S4 User inputs invalid email 1) App displays 'Error, please input valid email' 2) User returns to step 2) 02-AF-S4 User inputs invalid password 1) App displays 'Error, incorrect password' 2) User returns to step 3) 02-AF-S4 User inputs wrong password 5 times 1) App displays 'Account has been locked, please contact admin' 2) User returns to step 3) 02-AF-S4 User inputs invalid Google email a) App displays 'Error, please input valid Google email' b) User returns to step b) 02-AF-S4 User inputs invalid password a) App displays 'Error, incorrect password' b) User returns to step c)

	02-AF-S4 User inputs wrong password 5 times a) App displays 'Account has been locked, please contact admin' b) User returns to step c)
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

2. User must be able to login to the system manually or using their Google Account.
    - 2.1.1. The system must display text fields for the user to enter his information to login manually.
        - 2.1.1.1. The text fields must consist of Email.
        - 2.1.1.2. The text fields must consist of Password.
    - 2.1.2. The system must display text fields for the user to enter his information to login via the Google account.
        - 2.1.2.1. The text fields must consist of Google email.
        - 2.1.2.2. The text fields must consist of Google password.
    - 2.1.3. The user must fill in all of the text fields before clicking 'Sign In' button
        - 2.1.3.1. The system must verify the fields filled in by the user before logging the user into the system.
        - 2.1.3.2. The system must display an error message if the account does not exist.
    - 2.1.4. The system must log in the user if the information obtained from the text fields are verified.
        - 2.1.4.1. The email must be found in the database of the system.
            - 2.1.4.1.1. The password matches the corresponding email of the user.
        - 2.1.4.2. The Google account must be found in the database of the System.
    - 2.1.5. The user must be able to see the main menu interface upon login.

4.3 Logout

Use Case ID:	03		
Use Case Name:	Logout		
Created By:	Yi Heng	Last Updated By:	
Date Created:	30/01/2024	Date Last Updated:	

Actor:	User, System
Description:	User logs out of system
Preconditions:	User must be logged in to the App
Postconditions:	User is logged out of the App
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1) User opens main menu 2) User selected 'Log Out' 3) User is logged out of App 4) App returns to log in page
Alternative Flows:	NIL
Exceptions:	NIL
Includes:	Login
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

3. User must be able to logout from the system.
   3.1. The system main menu must display a button "Log Out" to exit the system.
      3.1.1. Upon clicking "Log Out", the user must be navigated back to the Login page.
   3.2. The user's information must still remain in the system for relogin.
   3.3. The system must display the same screen shown at the login page to the user upon Logout.

4.4 Search Carpark via Current Location

Use Case ID:	04		
Use Case Name:	Search Carpark via Current Location		
Created By:	Yi Heng	Last Updated By:	
Date Created:	30/01/2024	Date Last Updated:	

Actor:	User, Carpark API
Description:	User searches carparks nearby with current location
Preconditions:	User has logged into the App
Postconditions:	App displays nearby carparks
Priority:	High
Frequency of Use:	High
Flow of Events:	1) User selects ‘Carparks near me’ 2) Carpark API locates current location 3) Carpark API returns carpark locations nearby 4) App displays nearby carpark information
Alternative Flows:	04-AF-S3 Carpark API unable to find carparks nearby 1) App displays ‘unable to find carparks nearby’ 2) App returns to main menu
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

4. User must be able to search carpark via current location.
    4.1. The system main menu must display carparks near the users.
        4.1.1. The system must show the carparks nearby the user’s current location in a circle of 1 kilometer radius.
        4.1.2. Carparks displayed on the map must be clickable, allowing users to access more information
            4.1.2.1. Clicking on a carpark marker must lead to a page displaying detailed information.
                4.1.2.1.1. The detailed information must consist of Carpark name
                4.1.2.1.2. The detailed information must consist of Live availability
                4.1.2.1.3. The detailed information must consist of Distance
                4.1.2.1.4. The detailed information page must include a "Directions" button
                    4.1.2.1.4.1. Clicking the "Directions" button must initiate the process of providing directions
                    4.1.2.1.4.2. The system must display an outlined route from the user's current location to the selected carpark
                    4.1.2.1.4.3. The system must allow the user to cancel navigation at anytime by clicking the “X”.

4.5 View User Profile

Use Case ID:	05		
Use Case Name:	View User Profile		
Created By:	Shun Ping	Last Updated By:	Aaron
Date Created:	30/1/2024	Date Last Updated:	03/02/2024

Actor:	User, System
Description:	User view profile related to self
Preconditions:	User has logged into the App
Postconditions:	App displays information regarding user
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1) User selects 'View User Profile" 2) System navigates to page with all the information of the user 3) System display all user information 4) If the user selects the back button, the system returns to the selection screen. 5) If the user chooses to edit profile, user can choose between "Edit Name" or "Change Password". 6) If user chooses "Edit Name", then he uses the included use case Edit Name to edit the user name. 7) If user chooses "Change Password", then he uses the included use case Change Password to change the password. 8) System updates user profile based on option carried out.
Alternative Flows:	NIL
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

5. User must be able to view their current profile.
    5.1. The system must display a button "View User Profile" to view the user's profile.
        5.1.1. The system must display Name Registered.
        5.1.2. The system must display the Registered Email Address.
    5.2. User must be able to able to edit name.
    5.3. User must be able to change passwords.

4.6 Search Carpark via Search

Use Case ID:	06		
Use Case Name:	Search Carpark via Search		
Created By:	Yi Heng	Last Updated By:	Aaron
Date Created:	30/01/2024	Date Last Updated:	03/02/2024

Actor:	User, Carpark API
Description:	User searches carparks nearby with postal code or name
Preconditions:	User has logged into App
Postconditions:	App displays nearby carparks
Priority:	High
Frequency of Use:	High
Flow of Events:	1) User selects 'Search For Carparks' 2) User enters postal code or name of location 3) Carpark API searches for carparks nearby based on entered location 4) Carpark API returns carpark locations nearby 5) App displays nearby carpark information
Alternative Flows:	06-AF-S3 Carpark API unable to find carparks nearby 1) App displays 'unable to find carparks nearby' 2) Go to step 2)
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

6. User must be able to search carpark via search bar.
    6.1. The system main menu must display "Search" for user to enter his/her information.
        6.1.1. After selecting "Carparks via search bar", the system must display a text field for users to input their information.
            6.1.1.1. The text field must consist of "Enter Postal Code/ Enter Street Name"
                6.1.1.1.1. User must key in either postal code or street name to locate the carpark
                    6.1.1.1.1.1. The system must verify the information filled in the field.
                    6.1.1.1.1.2. The system must display "No results found. Try again" if not found.
                6.1.1.1.2. The system must show the carparks nearby the address in a circle of 1km radius if input is found on the map.
        6.1.2. Carparks displayed on the map must be clickable, allowing users to access more information
            6.1.2.1. Clicking on a carpark marker must lead to a page displaying detailed information.
                6.1.2.1.1. The detailed information must consist of Carpark name

6.1.2.1.2. The detailed information must consist of Live availability
6.1.2.1.3. The detailed information must consist of Distance
6.1.2.1.4. The detailed information page must include a "Directions" button
6.1.2.1.4.1. Clicking the "Directions" button must initiate the process of providing directions
6.1.2.1.4.2. The system must display an outlined route from the user's current location to the selected carpark
6.1.2.1.4.3. The system must allow the user to cancel navigation at anytime by clicking the “X”

4.7 Favourite Carpark

Use Case ID:	07		
Use Case Name:	Favourite Carpark		
Created By:	Yi Heng	Last Updated By:	Aaron
Date Created:	30/01/2024	Date Last Updated:	03/02/2024

Actor:	User, System, Carpark API
Description:	User favourites carparks
Preconditions:	User has logged into App
Postconditions:	Carparks has been favourited to their profile.
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1) App displays nearby carpark information after 'Search Carpark via Current Location' or 'Search Carpark via Search' 2) User clicks on carpark they wish to favourite 3) User clicks on 'Favourite Carpark' 4) App adds carpark to 'Favourite Carparks' under Favourite Carparks in main menu 5) App display "successfully favourited carpark"
Alternative Flows:	07-AF-S3 'Favourite Carparks' exceeds the limit of 5 1) App displays "Favourite Carparks list is full" 2) Go to step 1)
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

7. User must be able to favourite carparks in the app
    7.1. Clicking on a carpark marker on the map must lead to a page displaying detailed information.
        7.1.1. The detailed information page must include a "Favourite" button if the carpark does not exist in the User's list of favourite carparks.
            7.1.1.1. Clicking on "Favourite" button must add the carpark to the User's list of Favourite Carparks

4.8 Remove Favourite Carpark

Use Case ID:	08		
Use Case Name:	Remove Favourite Carpark		
Created By:	John	Last Updated By:	Aaron
Date Created:	30/01/2024	Date Last Updated:	03/02/2024

Actor:	User, System, Carpark API
Description:	User removes a favourited carpark (by favourite list or by map)
Preconditions:	User has logged into App, User has favourite carparks
Postconditions:	Favourited carpark has been removed from their profile
Priority:	Low
Frequency of Use:	Low
Flow of Events:	Favourite List 1) User clicks on 'View Favourite Carparks' in main menu 2) App displays list of favourite carparks 3) User clicks on a favourited carpark they wish to unfavourite 4) User clicks "Unfavourite Carpark" 5) App removes favourited carpark from user's favourite list 6) App displays successful removal of favourited carpark Search a) App displays nearby carpark information after 'Search Carpark via Current Location' or 'Search Carpark via Search' b) User clicks on a favourited carpark they wish to unfavourite c) User clicks on 'Unfavourite Carpark' d) App removes carpark from User profile e) App display successful removal of favourite carpark
Alternative Flows:	08-AF-S3 User has no favourite carpark 1) App displays 'No favourite carparks found' 2) Go to main menu
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

8. User must be able to remove favourited carparks in the app
    8.1. System must display carpark information after clicking on carpark on map.
        8.1.1. The detailed information page must include a "Unfavourite" button if the carpark already exists in the User's list of favourite carparks.
            8.1.1.1. Clicking on "Unfavourite" button must remove the carpark from the User's list of Favourite Carparks
    8.2. Clicking on favourited carpark in View Favourite Carparks in main menu must display favourited carparks information.
        8.2.1. The information page must include a "Unfavourite" button if the carpark already exists in the User's list of favourite carparks.

8.2.1.1. Clicking on “Unfavourite” button must remove the carpark from the User’s list of Favourite Carparks

4.9 View Favourited Carparks

Use Case ID:	09		
Use Case Name:	View Favourited Carparks		
Created By:	Yi Heng	Last Updated By:	Aaron
Date Created:	30/01/2024	Date Last Updated:	03/02/2024

Actor:	User, System, Carpark API
Description:	User view favourited carparks
Preconditions:	User has logged into App
Postconditions:	App displays list of favourited carparks
Priority:	High
Frequency of Use:	High
Flow of Events:	1) User clicks on 'View Favourite Carparks' in main menu 2) App displays list of favourite carparks 3) User clicks on a favourited carpark 4) App searches for carpark information in Carpark API 5) App displays carpark information of favourite carparks
Alternative Flows:	08-AF-S1 User has no favourite carpark 1) App displays 'No favourite carparks found' 2) Go to main menu
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

9. User must be able to view their favourite carparks.
    9.1. The system must have a "View Favourite Carparks" for the user to click on.
    9.2. The system must display the lists of favourite carparks once they clicked on the "View Favourite Carparks".
    9.3. The user must be able to select a carpark they have favourited before.
        9.3.1. The system must display carpark information about the selected favourite carpark.
            9.3.1.1. The information must consist of carpark availability.
            9.3.1.2. The information must consist of distance from current location.
            9.3.1.3. The information must consist of name of carpark saved as.
        9.3.2. The system must provide a "Directions" button next to each favourited carpark
            9.3.2.1. The system must initiate a navigation service to guide the user from their current location to the selected carpark upon clicking on the "Directions" button.
        9.3.3. The system must allow user to cancel navigation at any time.
            9.3.3.1. The system must provide a "X" button to cancel the current navigation.

4.10 Password Recovery

Use Case ID:	10		
Use Case Name:	Password Recovery		
Created By:	Aaron	Last Updated By:	Shun Ping
Date Created:	01/02/2024	Date Last Updated:	24/03/2024

Actor:	User, System
Description:	App resets user password
Preconditions:	User must already be registered in the application
Postconditions:	Password of the user in the database has been updated with the new password.
Priority:	High
Frequency of Use:	Low
Flow of Events:	1) User clicks on forgot password in the login page 2) App will take user to reset password page 3) App will prompt user for their email 4) User enters email 5) User clicks send email 6) System will send reset link to the email 7) User check email for link 8) User follow steps from email link 9) User enters new password 10) System changes password of user
Alternative Flows:	09-AF-S3 Email not in database 1) App outputs ‘invalid email’ 2) App returns to step 2
Exceptions:	NIL
Includes:	NIL
Special Requirements:	User is able to return back to login page at anytime
Assumptions:	User has internet connection
Notes and Issues:	NIL

10. User must be able to reset his/her password.
    10.1. The system must have a “Forgot password” on the login page.
        10.1.1. Upon clicking “Forgot password”, the user must be navigated to the reset password page.
            10.1.1.1. The system must display text fields for the user to enter his/her information to reset the password
                10.1.1.1.1. The text fields must consist of email.
                10.1.1.1.2. The system must display a send password link button.
    10.2. The user must fill in the email text field before clicking the send password link.
        10.2.1. The system will check the app database to see if an account with the email exists.
        10.2.2. The system will then send a reset password link to the email.
    10.3. The user must go to the email and click on the link.
    10.4. The user must fill in the new password.

10.5. The system must verify the fields when the user clicks the Reset Password button.
- 10.5.1.1. The new password must consist of at least 1 uppercase character.
- 10.5.1.2. The new password must consist of at least 1 lowercase character.
- 10.5.1.3. The new password must consist of at least 8 characters.
- 10.5.1.4. The new password must consist of at least 1 number.
- 10.5.1.5. The Confirm New Password must match the New Password.
- 10.5.1.6. The inputted OTP must match the OTP sent by the system to the user's email

10.5.2. The system must provide error messages describing the reason the user's password is rejected.

10.6. The system must reset the password upon successful verification.

.

4.11 Edit Name

Use Case ID:	11		
Use Case Name:	Edit Name		
Created By:	Shun Ping	Last Updated By:	
Date Created:	01/02/2024	Date Last Updated:	

Actor:	User, System
Description:	User change the name that is stored on the system.
Preconditions:	User has clicked on “View User Profile”
Postconditions:	Name of user in database has been updated with the new name.
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1) User clicks on “Edit Name” on the profile page 2) System displays the current “First name” and “Last name” to the user. 3) System prompts the user to enter the new first name and last name. 4) User enters the information and clicks on “Save”. 5) System stores and overwrites the new details into the database. 6) System changes the name of the user. 7) App will display the new name for the user.
Alternative Flows:	NIL
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

11. User must be able to edit their own name on the app.
    - 11.1. The system must display the user's current name as a reference.
    - 11.2. The system must display text fields for the user to enter the name they want to change to.
        - 11.2.1. The text field must consist of “First name”
        - 11.2.2. The text field must consist of “Last name”
    - 11.3. The system must have a “Save” button to confirm the new name.
    - 11.4. The system must have a “<-” button to return to the previous state.
    - 11.5. The system must verify the text field before updating the name of the user..
        - 11.5.1. The text fields must not be empty.
    - 11.6. The system must update the name upon successful verification.

4.12 Edit Favourite Carpark's Nickname

Use Case ID:	12		
Use Case Name:	Edit Favourite Carpark's Nickname		
Created By:	Shun Ping	Last Updated By:	
Date Created:	01/02/2024	Date Last Updated:	

Actor:	User, System, Carpark API
Description:	User edit the name of the carparks he/she favourited
Preconditions:	1) User has clicked on "View User Profile" 2) User has clicked on "View Favourited Carparks"
Postconditions:	Nicknames of carparks in database has been updated with the new value.
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1) User clicks on "View Favourited Carparks" on the main menu, then he uses the included use case View Favourite Carparks. 2) User selects the favourited carpark to rename. 3) User selects "Rename" from the pop up menu 4) System prompts the user to enter the new carpark's nickname 5) User enters the information and clicks on "Save". 6) System stores and overwrites the new details into the database. 7) System changes the nickname of the carpark for the user. 8) App will display the new nickname for the favourited carparks.
Alternative Flows:	NIL
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

12. User must be able to edit the nickname of favourited carparks.
    - 12.1. The system must provide a user interface that allows users to access and edit the nickname of their favourited carparks.
    - 12.2. The editing interface must display a list of favourited carparks.
        - 12.2.1. The interface must include the current nicknames.
        - 12.2.2. The interface must provide an option to edit each carparks.
    - 12.3. The system must allow users to modify the existing nickname.
        - 12.3.1. The system must offer a "Rename" button for each favourited carpark.
        - 12.3.2. The system must provide a text field within the interface for users to input the new nickname for the favourite carpark once "Rename" is pressed.
            - 12.3.2.1. The system must prompt the user to enter a nickname for the carpark.

12.3.2.2. The system must provide a button “Save” to update the nickname.
12.3.2.3. The system must provide a button “Cancel” to return to previous state.
12.4. The system must verify the text field before updating the nickname of the carpark.
12.4.1. The system must check the text field is not empty.
12.4.2. The system must check the nickname has not been used before.
12.5. The system must update the nickname upon successful verification.

4.13 Change Password

Use Case ID:	13		
Use Case Name:	Change Password		
Created By:	Shun Ping	Last Updated By:	
Date Created:	01/02/2024	Date Last Updated:	

Actor:	User, System
Description:	User change his/her current password
Preconditions:	1) User has clicked on “View User Profile” 2) User has clicked on “Change Password”
Postconditions:	Password of user in database has been updated with the new value.
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1) User clicks on “Change Password” on the profile page. 2) System prompts the user to enter current password and new password. 3) User enters the information and clicks on “Save”. 4) System checks whether the information satisfy the requirements and are valid. 5) System overwrites and stores the new password into the database. 6) App will display the “Password successfully changed”.
Alternative Flows:	13-AF-S3 Current password incorrect 1) System displays “Current password do not match database”. 2) System returns to Step 2. 13-AF-S3 If new password does not meet the requirements 1) System displays “Password does not meet requirements”. 2) System returns to Step 2.
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	User has internet connection
Notes and Issues:	NIL

13. User must be able to change password.
    - 13.1. The system must provide an interface for the user to change their password.
    - 13.2. The interface must display text fields for the user to enter the necessary information
        - 13.2.1. The text fields must consist of Current Password.
        - 13.2.2. The text fields must consist of New Password.
        - 13.2.3. The text fields must consist of Confirm New Password.
    - 13.3. The user must fill in all of the text fields (Current Password, New Password, Confirm New Password) before clicking the 'Save' button
        - 13.3.1. The system must verify the fields when the user clicks the ‘'Save’ button.

13.4. The system must verify the information filled in by the user before changing the password
  13.4.1. The current password entered must match the password on the database.
  13.4.2. The new password must meet the requirements.
    13.4.2.1. Old password must match user's current password
    13.4.2.2. The new password must consist of at least 1 uppercase character.
    13.4.2.3. The new password must consist of at least 1 lowercase character.
    13.4.2.4. The new password must consist of at least 8 characters.
    13.4.2.5. The new password must consist of at least 1 number.
    13.4.2.6. The new password must be different from the old password
    13.4.2.7. The confirm new password must match new password
  13.4.3. The system must provide error messages describing the reason the user's password is rejected.
13.5. The system must change the password upon successful verification.

5. Other Nonfunctional Requirements

Usability	1.	At least 80% of first time users must be able to enter a simple search query for a car park within 2 minutes after logging in.
	2.	At least 80% of new users should not take longer than 5 minutes to register for an account.
Reliability	1.	The app should be available and responsive, ensuring >99% uptime.
	2.	It should handle user inputs and requests reliably, with an error rate and crash probability not exceeding 1% under normal operating conditions.
	3.	Data synchronization between the app and server should be reliable and consistent, demonstrating a reliability rate of at least 99% across all synchronized datasets.
Supportability	1.	The app should be compatible with a range of mobile devices running on android, being compatible with at least 80% of android device models released in the past 5 years.
	2.	A comprehensive help and support system should be available within the app, providing guidance and assistance to users. At least 60% of users must be able to resolve their issue within 10 minutes of consulting the help and support system.
Scalability	1.	The app should be designed to handle a growing user base and increased traffic, with the capacity to accommodate a minimum of 10% annual user growth.
	2.	The app should perform efficiently even as the number of parked vehicles and users increases, demonstrating a response time of under 1000ms for standard user interactions, even with a concurrent user increase of up to 50% from current baselines.
Security	1.	User accounts should maintain a high level of security to prevent unauthorized access. Security testing should confirm that user accounts are resistant to common hacking techniques.
Performance	1.	The app should be responsive to user actions, maintaining a response time of under 1000ms for standard user interactions under normal operating conditions.
Maintainability	1.	Documentation for the app's codebase must be comprehensive, providing clear guidelines for code structure, dependencies, and system components.
	2.	The system should be modular, allowing for easier updates and modifications without impacting the entire application.
	3.	A version control system must be in place to track changes.

Appendix A: Glossary

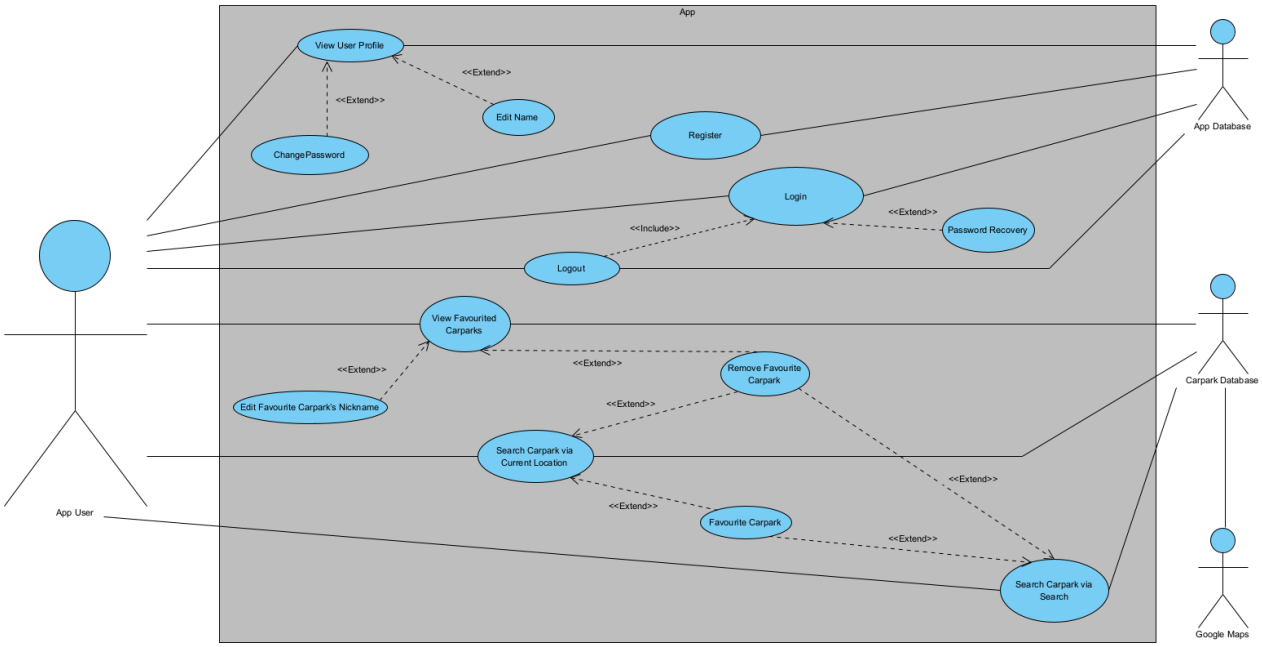
This segment will define all the terms, acronyms, abbreviations that are written in this SRS document.

Term	Definition
GPS	Global Positioning System. It is used to show the location of different places on the map.
Profile	The profile of a user, containing their personal information.
Login details	A combination of email and password a user requires to access their account.
User	A person who uses the app to look for carparks.
App	Our ParkWhere application
Map marker	An indication of where the carpark is, on the map.
API	Application Programming Interface, the carpark API used in our App is from data government.
Nearby	The distance of 2km from the current location
Carpark Information	Carpark rate, Availability, Distance from current location
User Information	User registered email and name
OTP	One-Time-Passsword. Used for verification when User forgets password

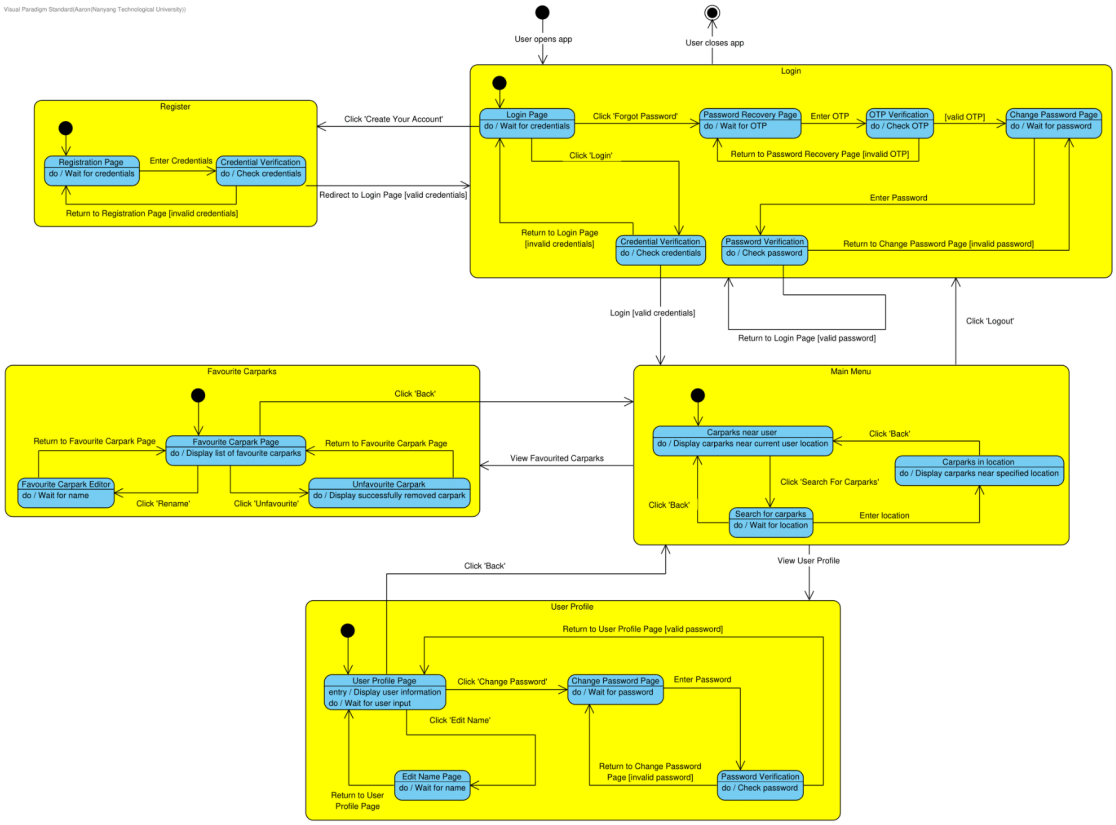
Appendix B: Analysis Models

This segment will consist of all analysis models. Clearer view of the picture are attached in the folder.

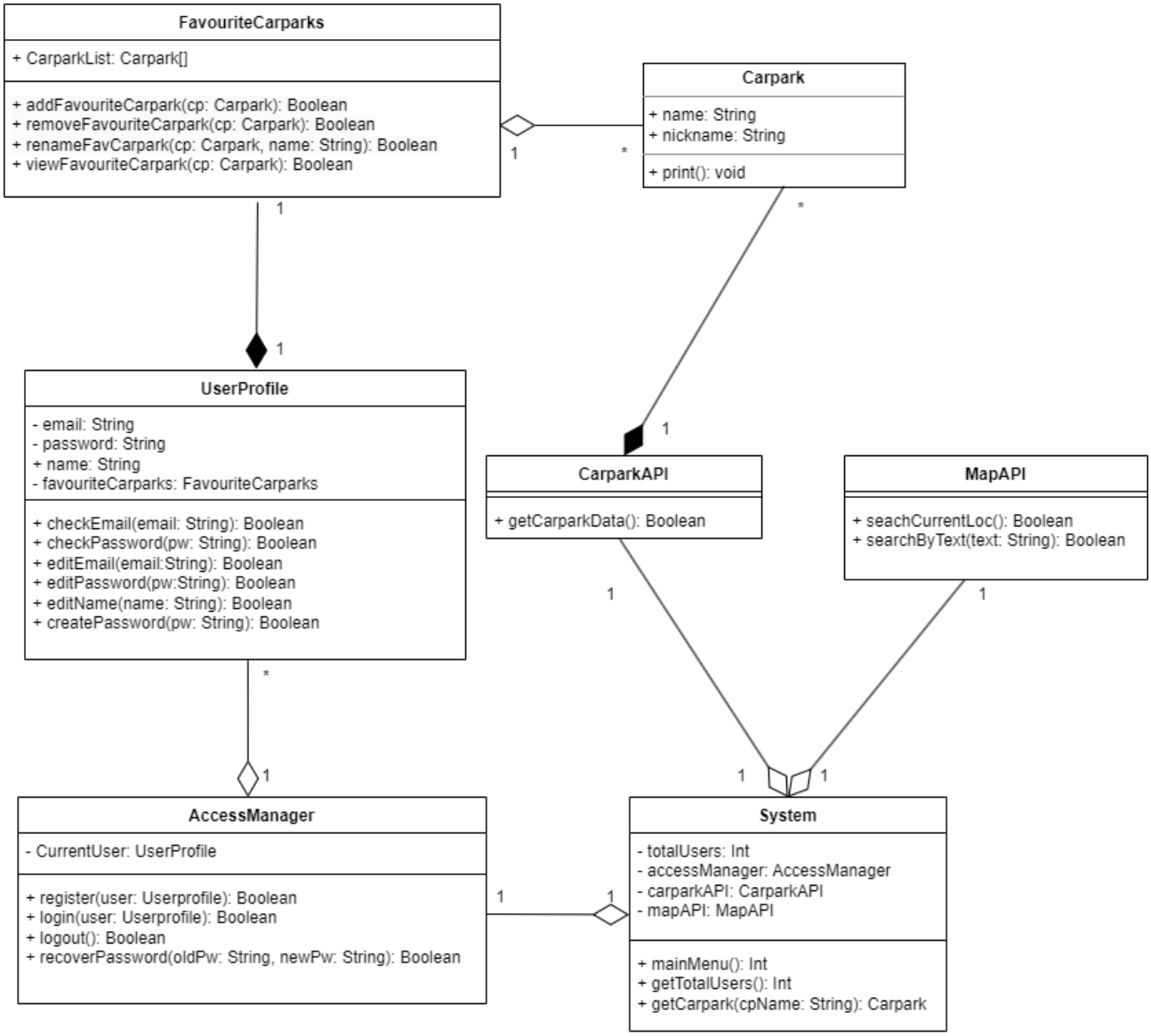
- Use Case Diagram



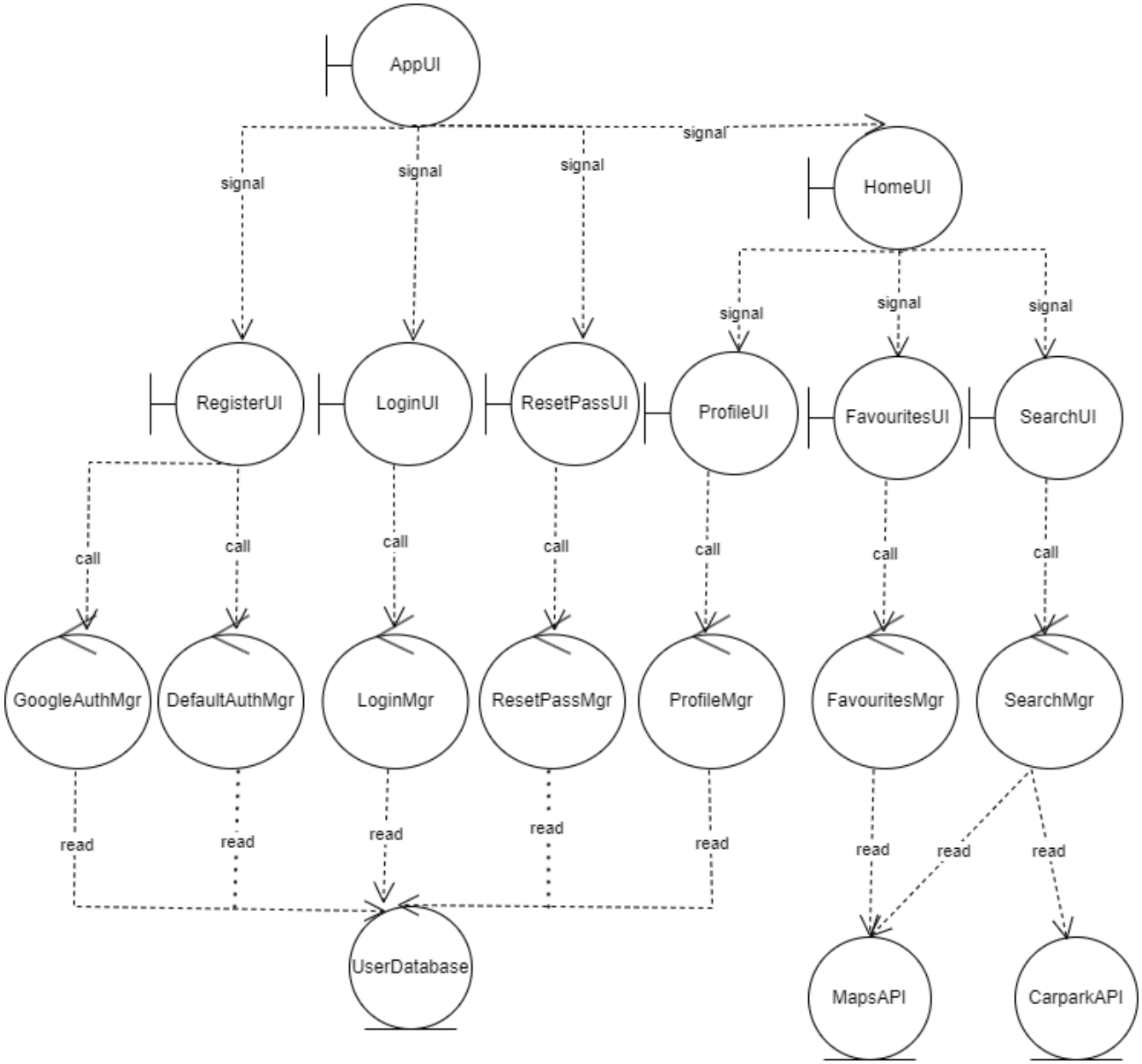
- Dialog Map (State Machine Diagram)



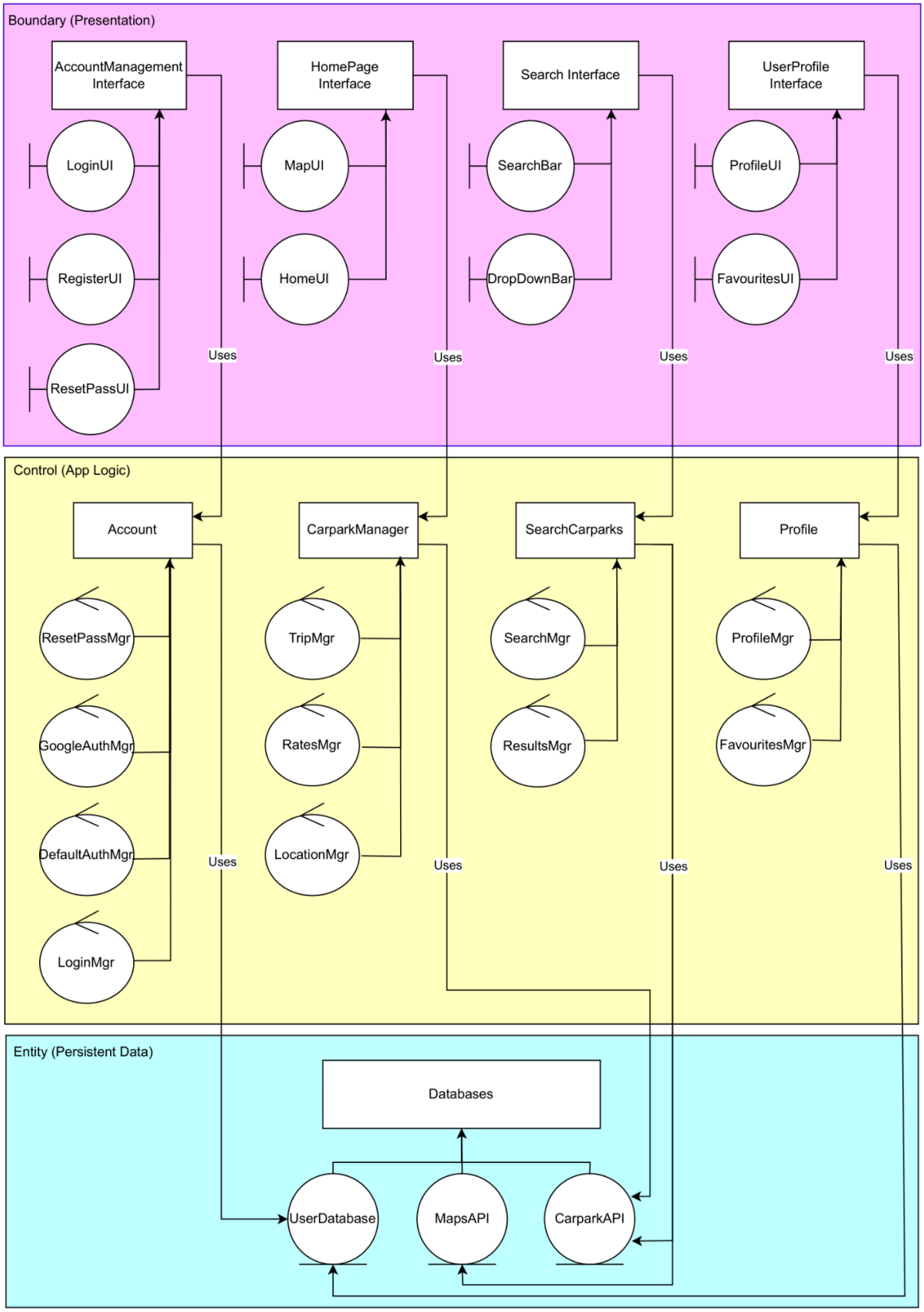
- UML Class Diagram



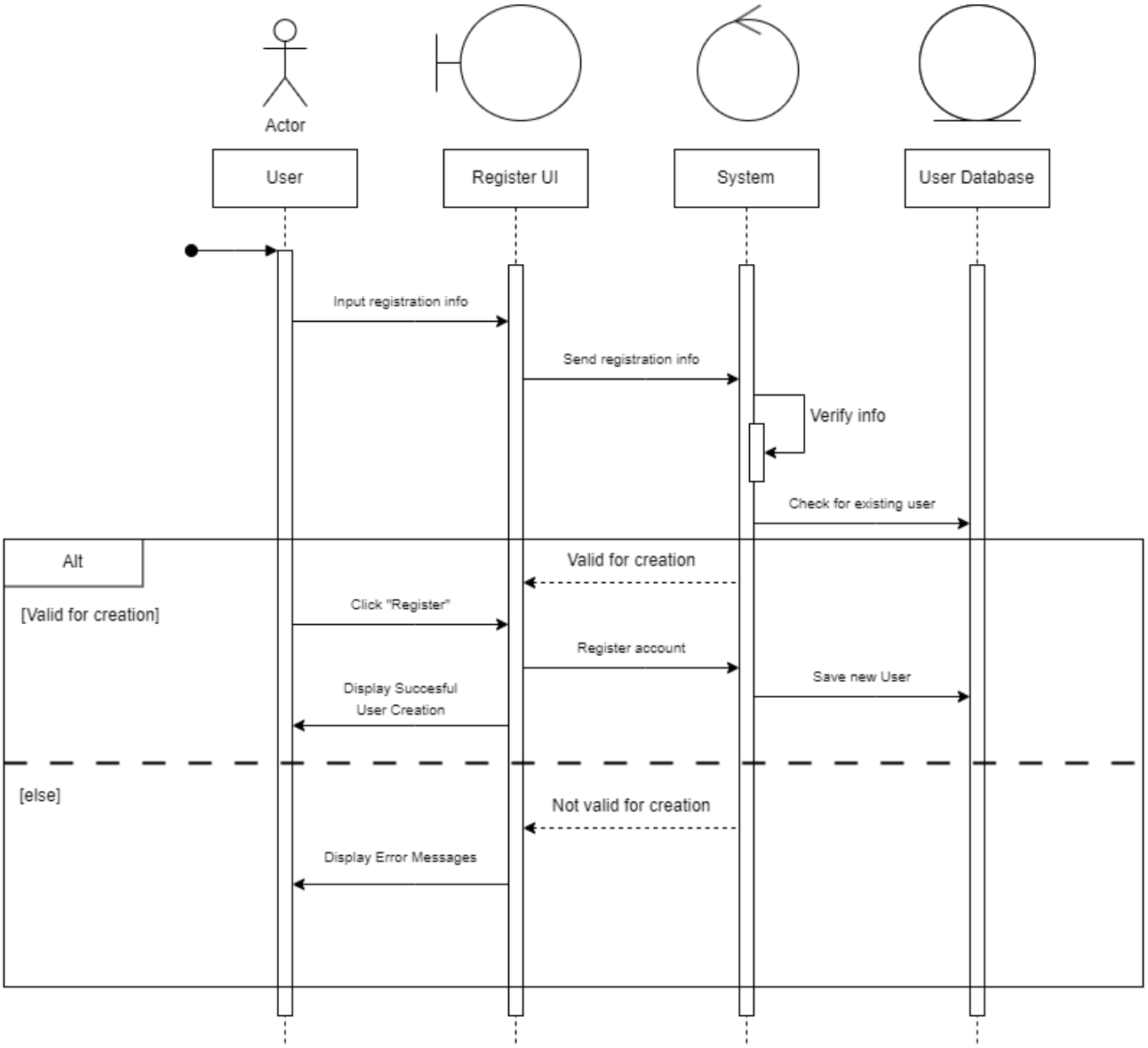
- Entity-Control-Boundary Diagram



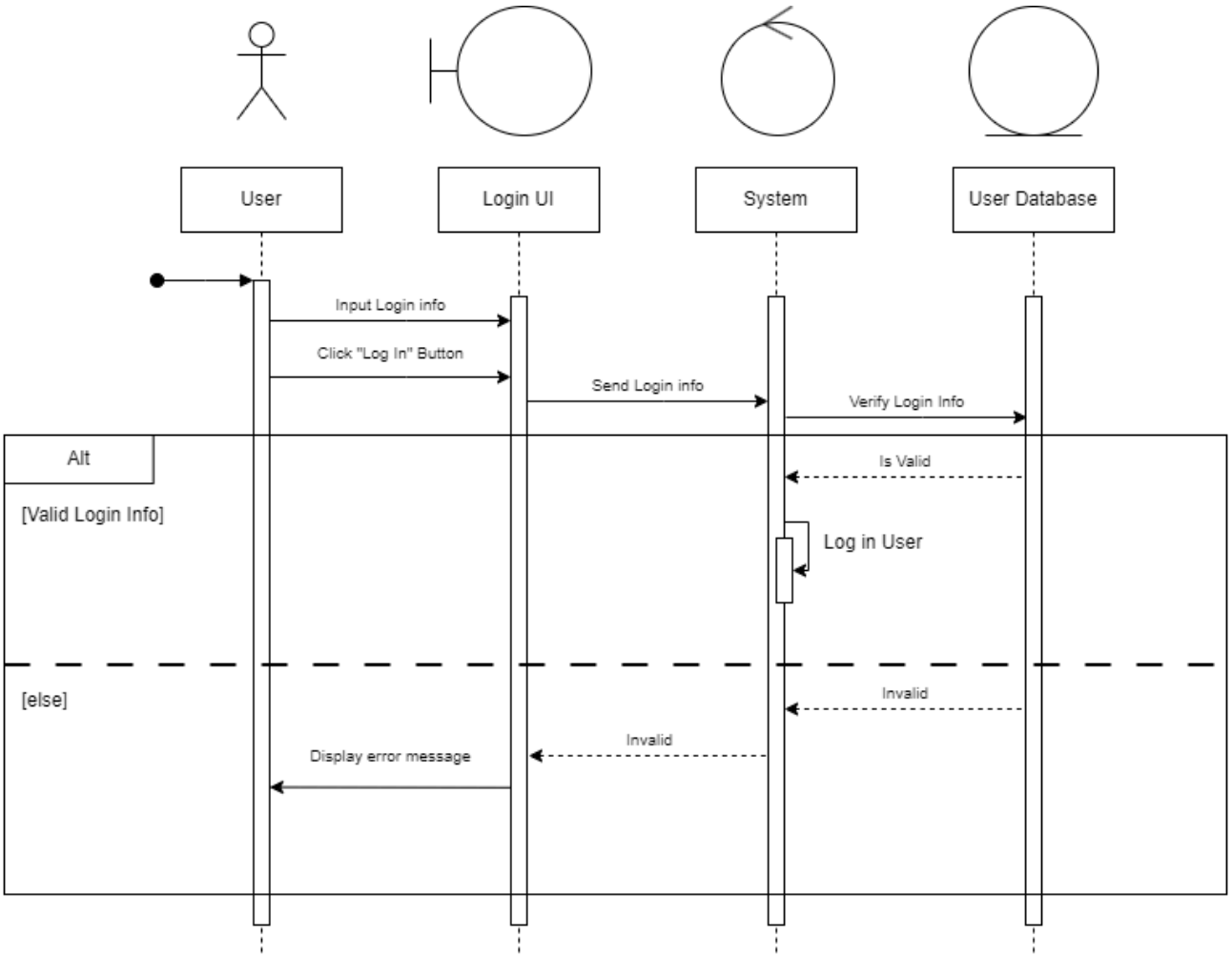
- Software Architecture (3-Layered Architecture)



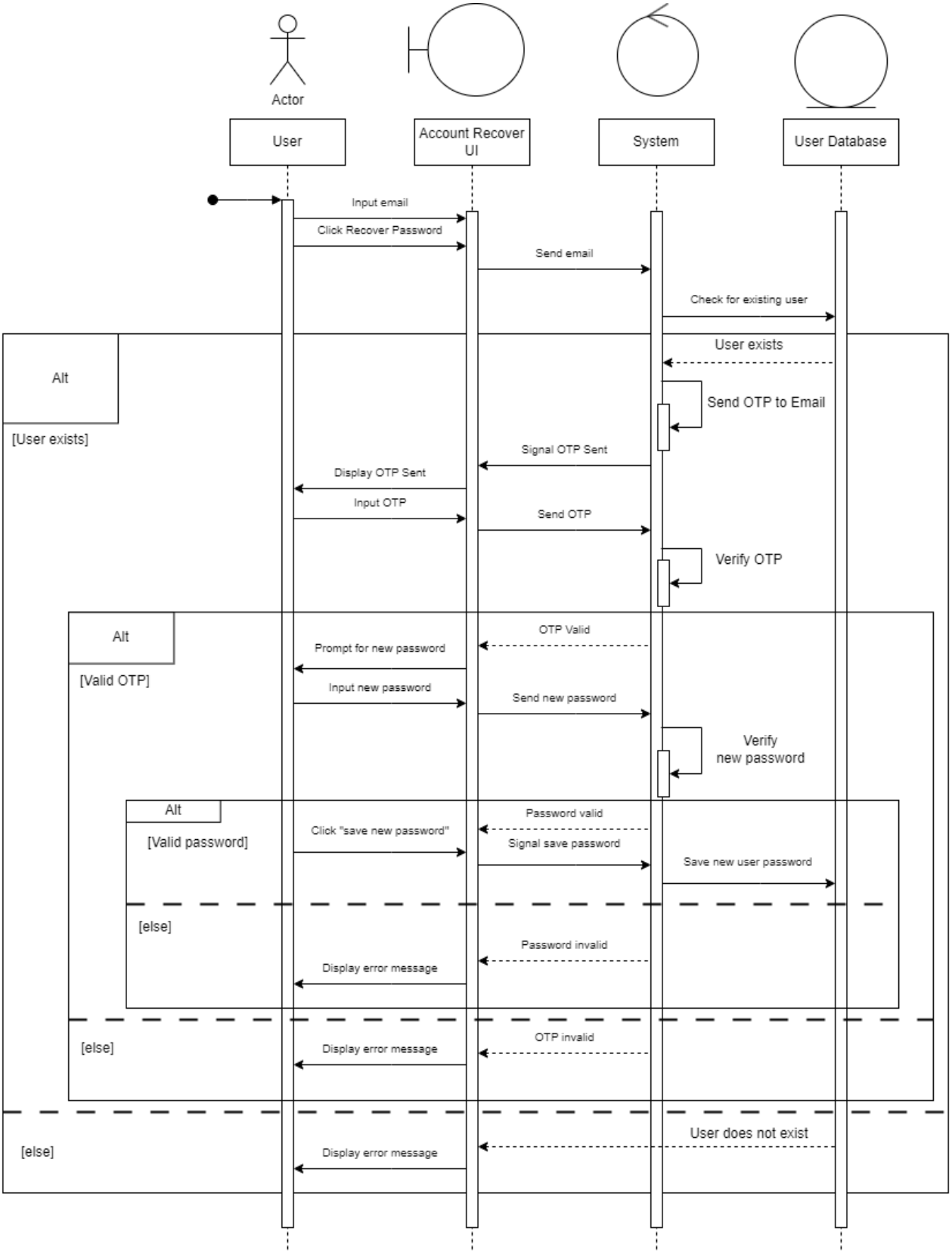
- Sequence Diagram
  - Register



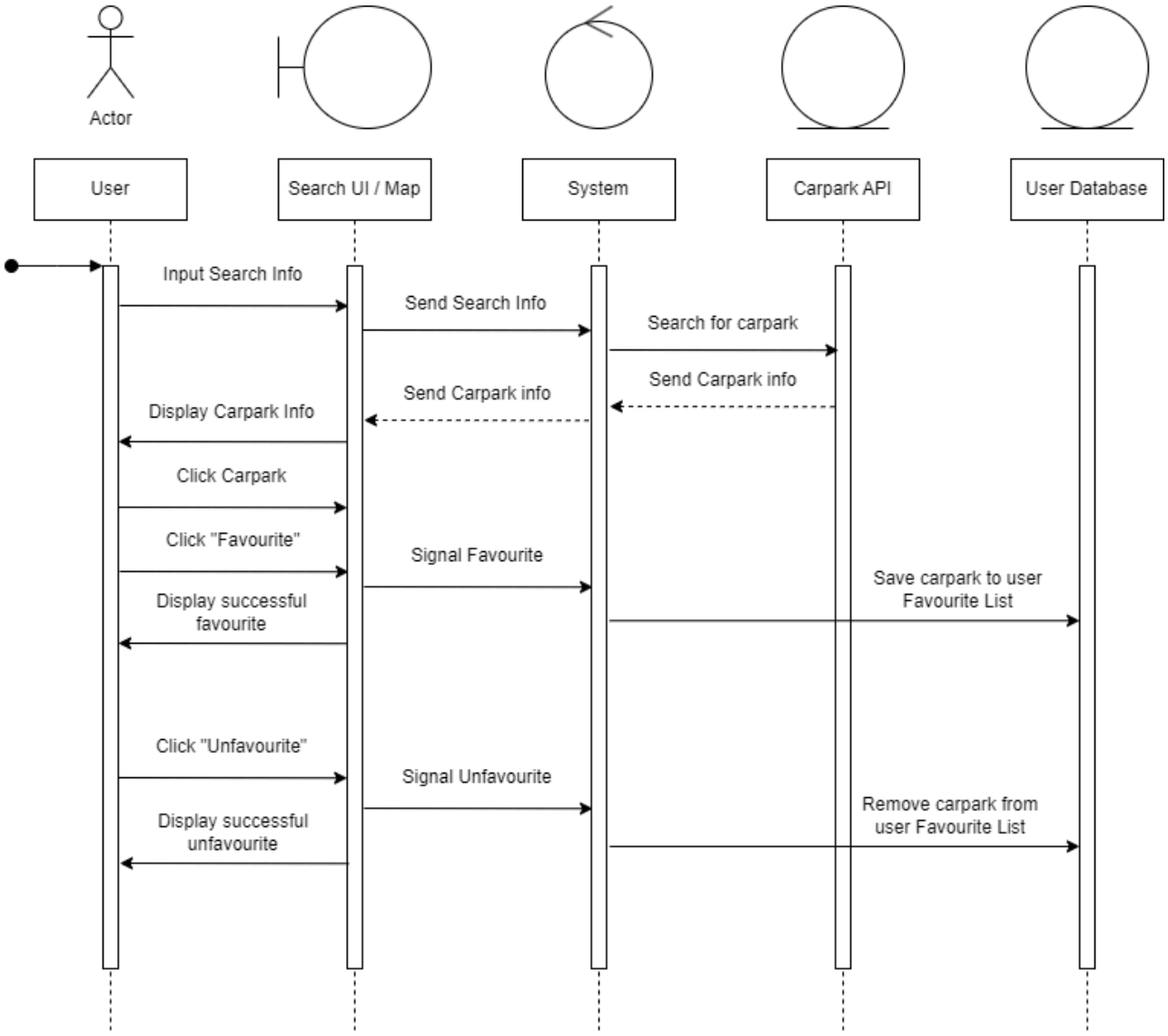
- Login


User
Login UI
System
User Database
Input Login info
Click "Log In" Button
Send Login info
Verify Login Info
Alt
[Valid Login Info]
Is Valid
Log in User
[else]
Invalid
Invalid
Display error message

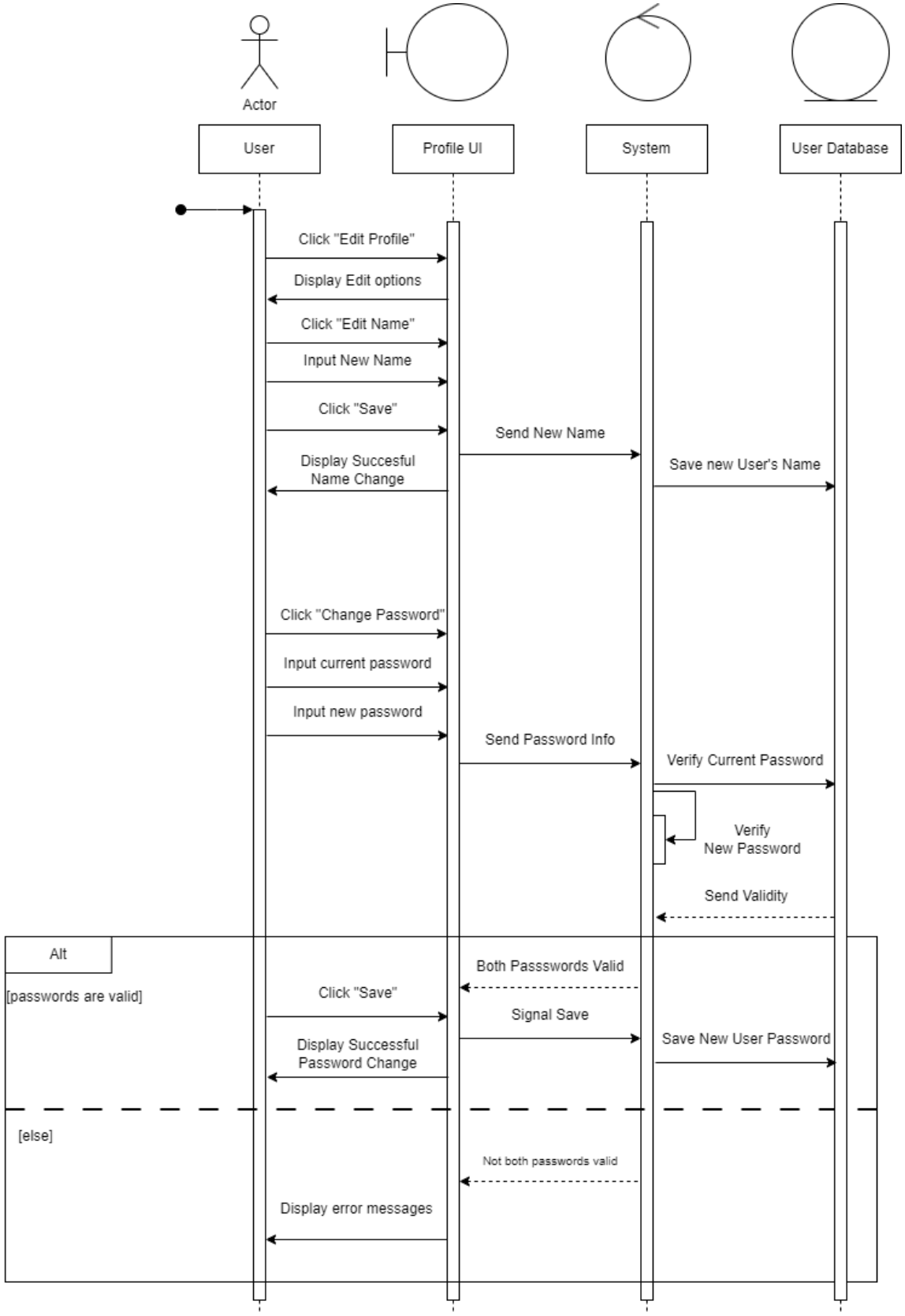
- Recover Account



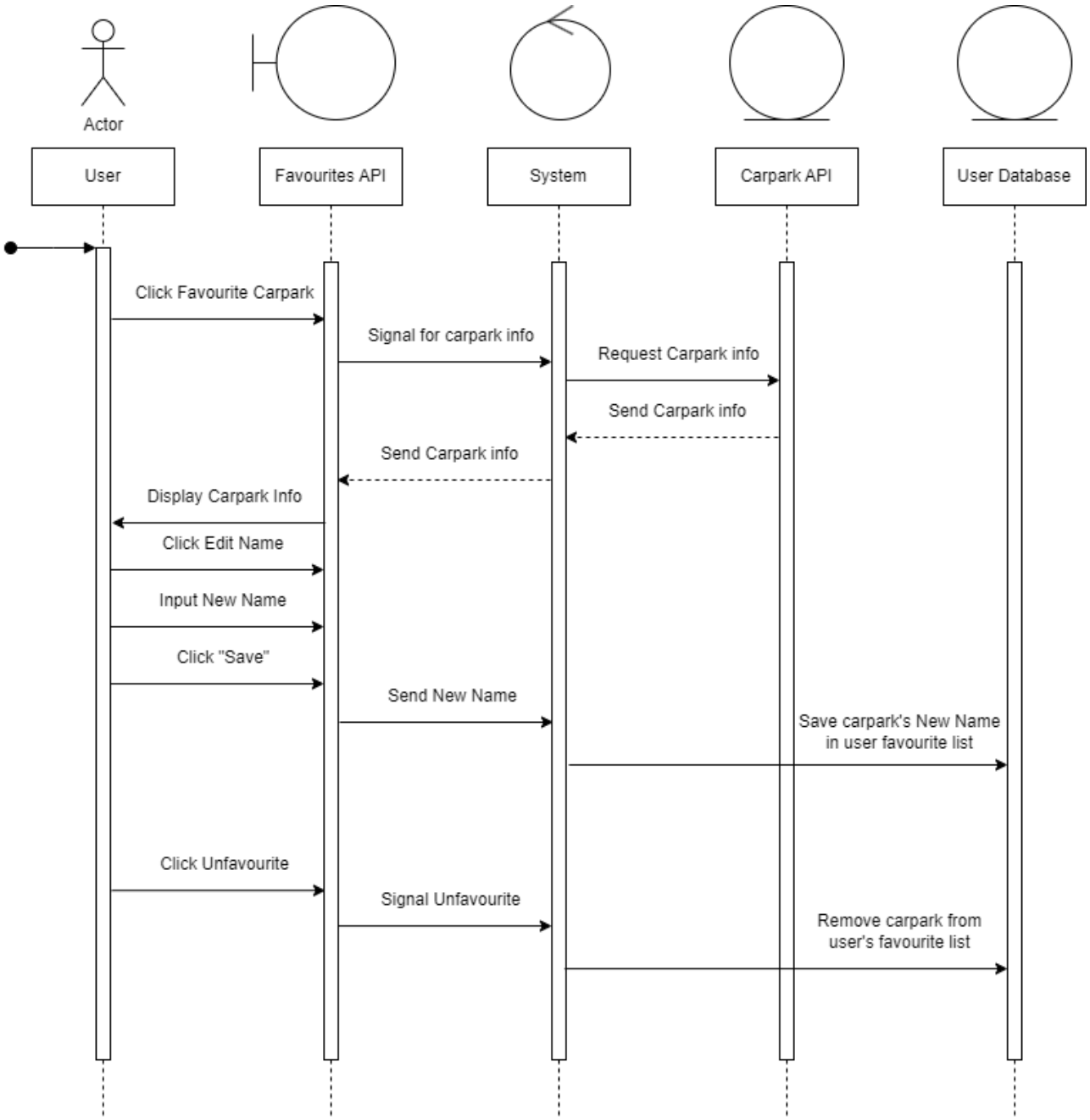
- Search


Actor
User
Search UI / Map
System
Carpark API
User Database
Input Search Info
Send Search Info
Search for carpark
Send Carpark info
Send Carpark info
Display Carpark Info
Click Carpark
Click "Favourite"
Signal Favourite
Save carpark to user Favourite List
Display successful favourite
Click "Unfavourite"
Signal Unfavourite
Remove carpark from user Favourite List
Display successful unfavourite

- Profile


Actor
User
Profile UI
System
User Database
Click "Edit Profile"
Display Edit options
Click "Edit Name"
Input New Name
Click "Save"
Send New Name
Display Succesful Name Change
Save new User's Name
Click "Change Password"
Input current password
Input new password
Send Password Info
Verify Current Password
Verify New Password
Send Validity
Alt
[passwords are valid]
Both Passswords Valid
Click "Save"
Signal Save
Save New User Password
Display Successful Password Change
[else]
Not both passwords valid
Display error messages

- Favourites


Actor
User
Favourites API
System
Carpark API
User Database
Click Favourite Carpark
Signal for carpark info
Request Carpark info
Send Carpark info
Send Carpark info
Display Carpark Info
Click Edit Name
Input New Name
Click "Save"
Send New Name
Save carpark's New Name in user favourite list
Click Unfavourite
Signal Unfavourite
Remove carpark from user's favourite list