



## **Software Engineering**

### **Final Report (ParkingSlot)**

Submitted by:

Tan Jia Wei U1821234H

Wong Chun Foong U1822939L

Goh Jun Le U1820246G

An Zhi Andrew Cai N1902005C

Emmi Mannonen N1902011B

Nurul Sabrina Binte Mohammed Riduwan U1822849A

# **Report Outline**

## **1. Software Requirements Specifications**

- 1.1. Product Description
  - 1.1.1. Purpose of System
  - 1.1.2. Scope of System
  - 1.1.3. User and Stakeholders
  - 1.1.4. Assumptions and Constraints
- 1.2. Functional Requirements
- 1.3. Non-functional Requirements
- 1.4. Interface Requirements
  - 1.4.1. User Interface
  - 1.4.2. Software Interface
  - 1.4.3. Hardware Interface
- 1.5. Data Dictionary

## **2. Project Management and Scheduling**

- 2.1. Gantt Chart
- 2.2. Network Diagram

## **3. Use Case Model**

- 3.1. Use Case Diagram
- 3.2. Use Case Description

## **4. Class Diagram**

## **5. Sequence Diagrams**

## **6. State Machine Diagram**

## **7. Application Architecture**

## **8. Testing**

- 8.1. Blackbox Testing
- 8.2. Whitebox Testing

## 1. **Software Requirements Specifications**

### 1.1. **Product Description**

#### 1.1.1. **Purpose of System**

ParkingSlot aims to solve daily frustrating problems of motorists in Singapore. Motorists often spend time wasting in car parks due to a lack of parking space as they do not have information upfront about the car park availability. In addition, they have uncertainty about parking charges that they will incur, as they often do not know where to check the parking rates, or find it a hassle to do so. All in all, ParkingSlot serves to be a solution to these problems by serving as a one-stop portal for carpark rates and availability information.

#### 1.1.2. **Scope of System**

ParkingSlot is designed as a progressive web application that can run on Desktop, Android and iOS devices.

#### 1.1.3. **User and Stakeholders**

The users are motorists who have either a Car, Motorcycle or Heavy Vehicles. Stakeholders include partners who enable the system to work in production. GovTech is an important partner as they provide us with the data of real-time availability and parking rates through APIs. Lastly, stakeholders include the project developers who are in charge of the development, maintenance and upgrading of the applications.

#### 1.1.4. **Assumptions and Constraints**

ParkingSlot assumes that any user can run this application on his device, and an internet connection is required to fetch the latest data regarding real-time availability and carpark rates. Constraints of this application include the limitations of the public APIs provided by GovTech. There is a lot of duplicated car park information, and there is insufficient data regarding information like carpark rates and real-time availability for some car parks. In addition, there are dependencies on GovTech APIs to fetch real-time data from and also Google Maps APIs to provide routes and directions.

## **1.2. Functional Requirements**

### **1.2.1. Register/Login**

- 1.2.1.1. The motorist shall be able to sign up for an account
  - 1.2.1.1.1. The motorist must input their first name, last name, username, email, phone number, password and confirm password
  - 1.2.1.1.2. The backend system must check if the username and email are not registered. If it is, return the error message back
- 1.2.1.2. The motorist shall be able to login to his account
  - 1.2.1.2.1. The backend system must check if the username and email is registered. If it is, return the error message back
  - 1.2.1.2.2. If the username and password is valid, the backend system shall return a success status code and the frontend system will redirect the motorist to the home page
  - 1.2.1.2.3. If the username or password is invalid, the backend system will return an error message back and deny access to the motorist
- 1.2.1.3. The frontend application must check that the password and confirm password match otherwise display message
- 1.2.1.4. The frontend application must check that all fields are validated before submitting to the backend
- 1.2.1.5. The frontend application and backend system must be able to check if the password met it's required strength. (8 characters total, 1 uppercase and 1 special character and 1 digit)

### **1.2.2. Home**

- 1.2.2.1. Once logged in, the backend system must return 20 records of the car park information in ascending order using carpark name.
- 1.2.2.2. Basic fields returned will be the carparkname, address and availability
- 1.2.2.3. The system must return an individual car park information when requested
  - 1.2.2.3.1. Additional fields returned will be carpark id, carpark name, agency type, address, total available lots, total lots, car

availability, car capacity and carpark rates for car, motorcycle and heavy vehicle

1.2.2.4. The frontend application must allow motorist to query

1.2.2.4.1. The motorist must be able to filter by ascending order

1.2.2.4.2. The motorist must be able to filter by electronic parking

1.2.2.4.3. The motorist must be able to filter by central area

1.2.2.4.4. The motorist must be able to filter by location and range

1.2.2.4.5. The motorist must be able to filter by vehicle type

1.2.2.4.6. The motorist must be able to filter by agency type

1.2.2.5. The backend system must return the calculated cost based on user input

1.2.2.5.1. The motorist must be able to get calculated cost by number of hours, car park type and date/time

1.2.2.5.2. If there is no price available, a response will be returned to the motorist

### 1.2.3. **Maps**

1.2.3.1. The frontend application must be able to locate the motorist current location

1.2.3.2. The frontend application must be able to display maps, car park markers and the motorist current location marker

1.2.3.3. The frontend application must be able to return the directions to go from motorist current location to the specified destination

### 1.2.4. **Favorites**

1.2.4.1. The system must fetch the list of car parks that the motorist had favorited

1.2.4.2. The system must be able to delete car park from the motorist favorites list

### 1.2.5. **Profile**

1.2.5.1. The system must be able to fetch the user information

1.2.5.1.1. Motorist data display must consist of motorist first name

1.2.5.1.2. Motorist data display must consist of motorist first name

- 1.2.5.1.3. Motorist data display must consist of motorist last name
- 1.2.5.1.4. Motorist data display must consist of motorist email
- 1.2.5.1.5. Motorist data display must consist of motorist contact number
- 1.2.5.1.6. Motorist data display must consist of motorist username
- 1.2.5.2. The motorist shall be able to change his/her account password if needed.
  - 1.2.5.2.1. The motorist must input their current password, a new password and confirmation password
  - 1.2.5.2.2. The system shall reset the motorist's password link to the motorist's email address

#### 1.2.6. **Admin**

- 1.2.6.1. The administrator is able to view all motorists
- 1.2.6.2. The administrator is able to create new motorist
- 1.2.6.3. The administrator is able to edit motorist
- 1.2.6.4. The administrator is able to delete motorist

#### 1.2.7. **Feedback**

- 1.2.7.1. The administrator is able to view all feedback
- 1.2.7.2. The administrator is able to reply to the feedback and approve it once resolved
- 1.2.7.3. The administrator is able to revoke a feedback that have been approved

### 1.3. **Non-functional Requirements**

#### 1.3.1. **Usability**

- 1.3.1.1. The system functions must be organized such that the motorist will not have any problem navigating through the application
- 1.3.1.2. Errors or exception messages must be handled to eliminate any code warning messages to be shown to the motorist.
- 1.3.1.3. Error messages must be user friendly and understood by the normal motorist.

### 1.3.2. **Performance**

- 1.3.2.1. System must be able to handle a sudden surge of new users no more than 10,000
- 1.3.2.2. System must be able to provide timely feedback of no more than 1 seconds to guide the motorist through errors
- 1.3.2.3. System must return a response within 2 seconds upon motorist interaction

### 1.3.3. **Reliability**

- 1.3.3.1. System must be available to motorist at least 99.9% of the time from 7am to 12am
- 1.3.3.2. System must be able to handle a sudden surge of new users no more than 10,000
- 1.3.3.3. After restarting, full system functionality must be restored in less than 5 minutes
- 1.3.3.4. Upon entering the application again after exiting, the motorist previous carpark data must still be intact
- 1.3.3.5. There must be monthly backups for disaster recovery in case the database gets wiped out due to corruption
- 1.3.3.6. 99.9% of the systems failure must be repairable in an hour

### 1.3.4. **Security**

- 1.3.4.1. System shall encrypt the motorist password upon registration and no plaintext are to be stored in the database
- 1.3.4.2. System must be able to track every motorist login date and time and in addition display the last access date upon sign on
- 1.3.4.3. System shall be able to set the number of login attempts before locking the motorist account
- 1.3.4.4. System shall be able to enforce a certain password requirement
- 1.3.4.5. The website/application and database connections must be secured with encryption to prevent either system from getting compromised

1.3.4.6. The system must be able to detect denial-of-service (DoS) attacks

1.3.5. **Supportability**

1.3.5.1. The website/application shall be able to be deployed on any cloud server such as Amazon Web Services, Azure, HostGator, Dream host etc.

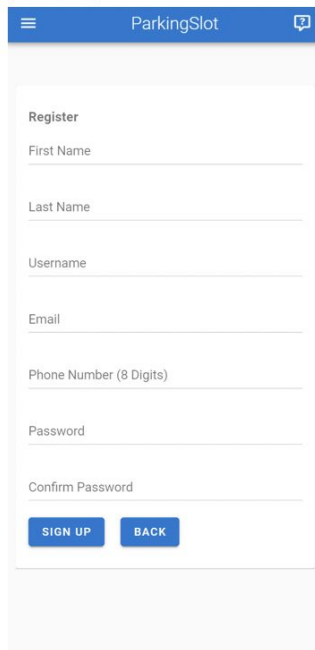
1.3.5.2. The database must be able to migrate to any other commercial products that support standard SQL queries



## 1.4. Interface Requirements

### 1.4.1. User Interfaces

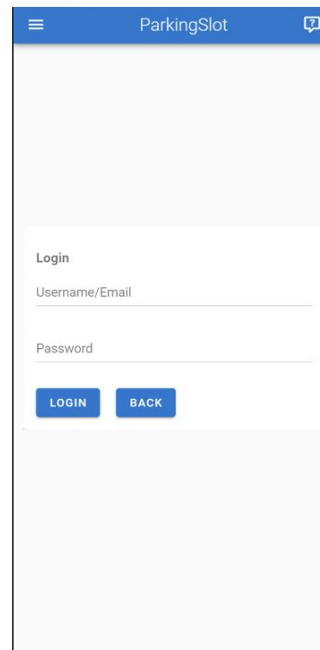
#### Register



The Register form is displayed within a mobile app interface titled 'ParkingSlot'. It features a list of input fields for user registration: First Name, Last Name, Username, Email, Phone Number (8 Digits), Password, and Confirm Password. At the bottom of the form are two buttons: 'SIGN UP' and 'BACK'.

The motorist must be able to register for an account on this page

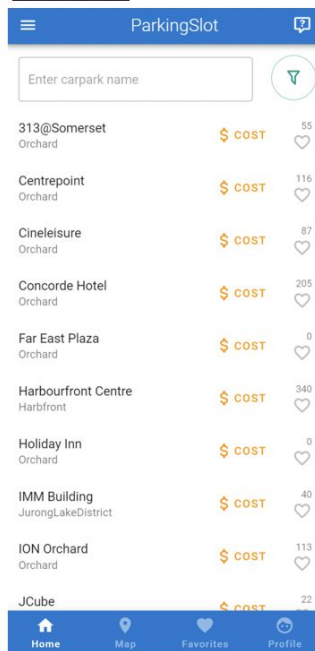
#### Login



The Login form is displayed within a mobile app interface titled 'ParkingSlot'. It features two input fields: Username/Email and Password. Below the fields are two buttons: 'LOGIN' and 'BACK'.

The motorist must be able to enter his username and password to login here

#### Homepage



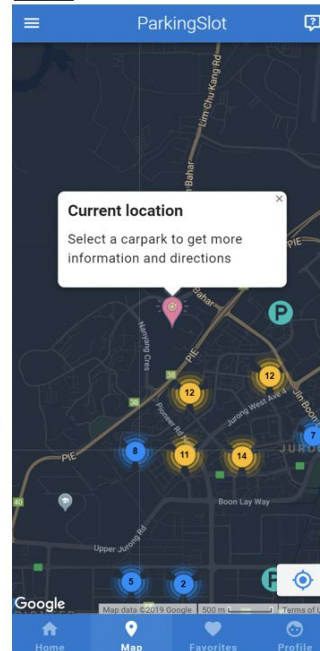
The Homepage interface shows a list of carpark entries. Each entry includes the carpark name, location, cost, and a heart icon for favorites. The list is as follows:

Carpark Name	Location	Cost	Favorites
313@Somerset	Orchard	\$ COST	55
Centrepont	Orchard	\$ COST	116
Cineleisure	Orchard	\$ COST	87
Concorde Hotel	Orchard	\$ COST	205
Far East Plaza	Orchard	\$ COST	0
Harbourfront Centre	Harbfront	\$ COST	340
Holiday Inn	Orchard	\$ COST	0
IMM Building	JurongLakeDistrict	\$ COST	40
ION Orchard	Orchard	\$ COST	113
JCube		\$ COST	22

At the bottom of the screen is a navigation bar with four icons: Home, Map, Favorites, and Profile.

The motorist must be able to view carpark information, search, filter, calculate cost and add carpark to favorites list.

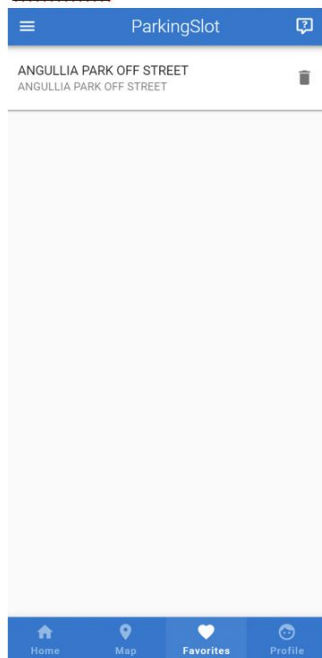
#### Maps



The Maps interface shows a map view of the carpark locations. A pop-up window titled 'Current location' is displayed, with the text 'Select a carpark to get more information and directions'. The map shows various carpark markers and a navigation bar at the bottom with icons for Home, Map, Favorites, and Profile.

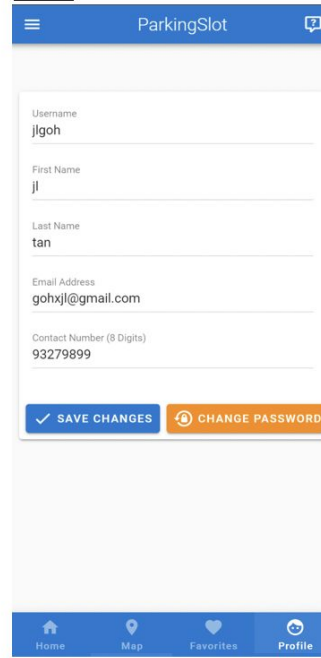
The motorist must be able to view carpark markers, view real-time availability of lots and check routes and directions.

### Favorites



The motorist must be able to view carpark information on his favourites list, and remove them from his favourites list.

### Maps



The motorist must be able to update his profile information and also change password.

#### 1.4.2. Software Interfaces

- Public APIs from data.gov.sg
  - LTA API for LTA Carpark Data
  - URA API for URA Carpark Data
  - HDB API for HDB Carpark Data
- Google Maps API
  - Directions API to calculate directions between locations
  - Maps Javascript API to customize maps with carpark markers and display on web pages and mobile devices
  - Places API to return location information from HTTP requests
- ASP.NET Core Web API MVC
  - SQL Database for data persistence

### 1.4.3. **Hardware Interface**

#### 1.4.3.1. Azure Production Server S1

1.4.3.1.1. Windows Server

1.4.3.1.2. 100 Total ACU (Compute unit)

1.4.3.1.3. 1.75 GB Memory

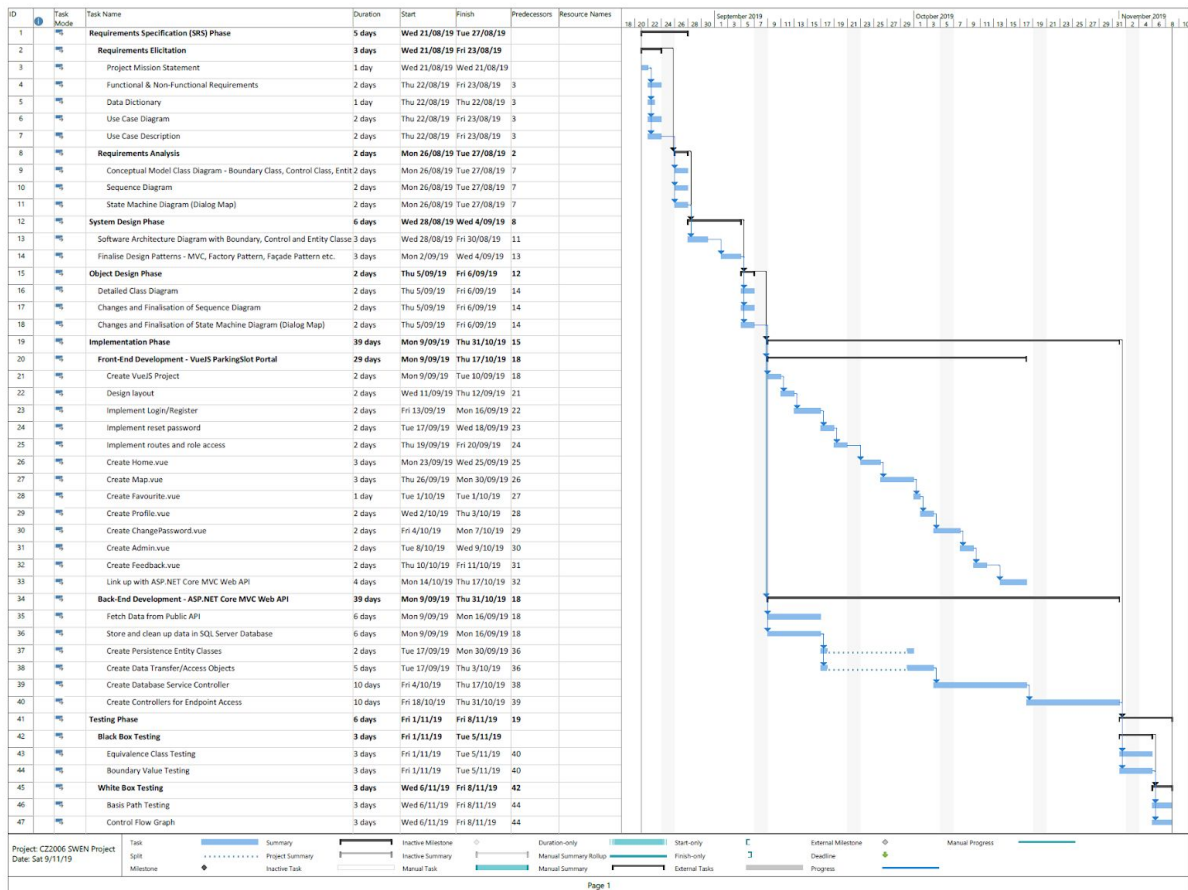
1.4.3.1.4. A-Series compute equivalent

## 1.5 Data Dictionary

Term	Definition
Motorist	Someone who drives a car, motorcycle or heavy vehicle
Vehicle	A mode of transporting people or goods, especially on land, such as a car, motorcycle or heavy vehicle
Multi Story Carpark	A large building with several floors for parking vehicles in. Commonly found nearby HDB flats
Parking Lot	A dedicated area that is intended for parking vehicles
Car park Availability	The real-time number of available lots in the car park
Parking Costs	The amount of parking costs that a motorist incur with respect the parking hours spent and parking rates charged
HDB Carpark	Car parks for Housing Development Board flats which are subsidised public housing for Singaporeans
URA Carpark	Car parks managed by Urban Redevelopment Authority which are outside of HDB estates
LTA Carpark Data	The LTA carpark data consist of major shopping malls and developments within Orchard, Marina, HarbourFront, Jurong Lake District.
Map	A representation of Singapore land showing buildings, cities, roads, car parks and etc

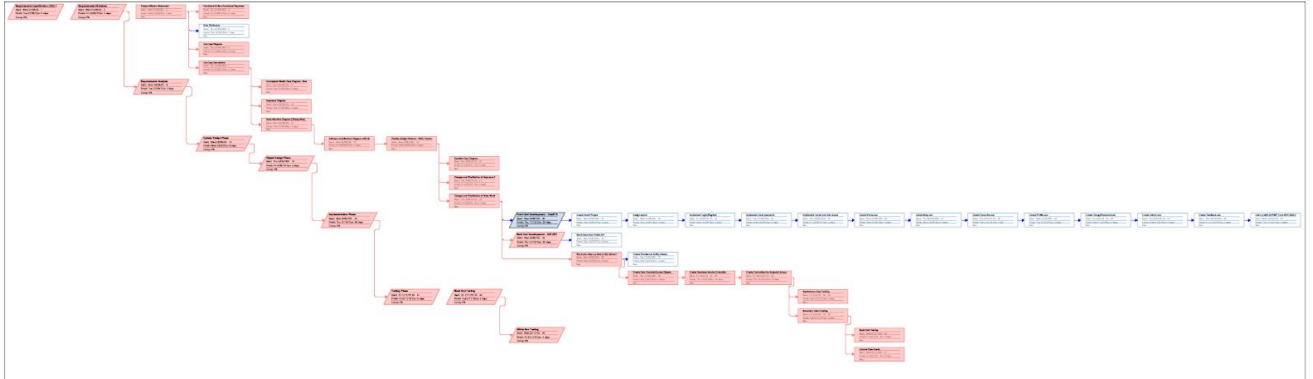
## 2. Project Management and Scheduling

### 2.1. Gantt Chart



Please refer to the full Gantt Chart attached in SVN for better resolution.

## 2.2. Network Chart

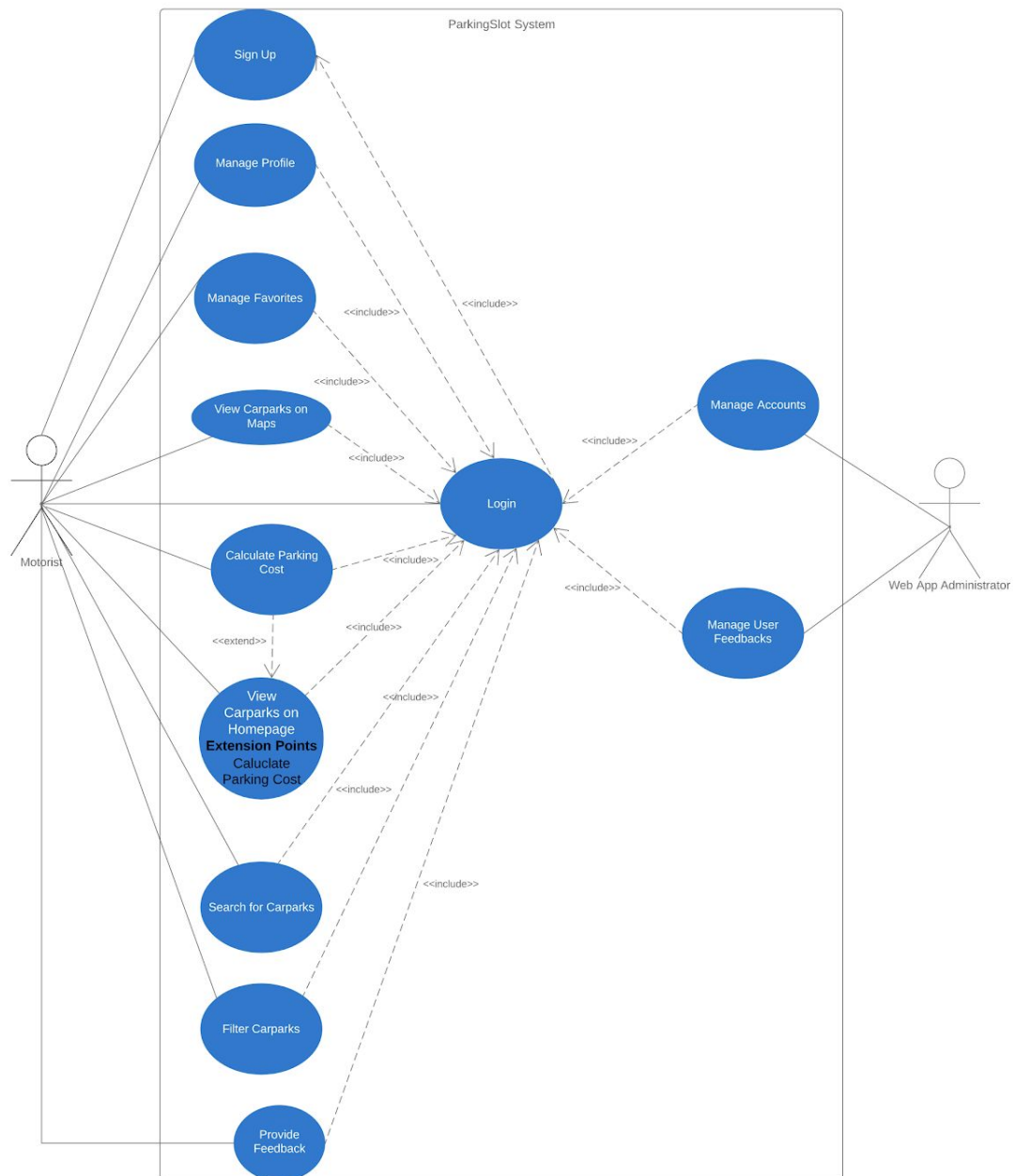


Please refer to the network diagram attached in SVN for better resolution.

### 3. Use Case Model

#### 3.1. Use Case Diagram

UML Use Case Diagram



### 3.2. Use Case Description

Use Case ID:	001		
Use Case Name:	<b>Sign Up</b>		
Created By:	Chun Foong	Last Updated By:	Jun Le
Date Created:	21/8/2019	Date Last Updated:	9/11/2019

Actor:	Motorist
Description:	Motorist registers account
Preconditions:	Motorist does not have the account
Postconditions:	Motorist successfully creates account
Priority:	High
Frequency of Use:	Motorist will only do this once
Flow of Events:	<ol style="list-style-type: none"><li>1. Motorist key in First Name, Last Name, Username, Email, Phone Number, Password, Confirm Password</li><li>2. System verifies Motorist details</li><li>3. If valid, Motorist able to proceed to login page</li></ol>
Alternative Flows:	<p>Motorist application invalid if email address, username and phone number is taken: Error message: "Field is already taken"</p> <p>Motorist application invalid if Password is not at least 8 characters long and does not have at least 3 of 4 following: Uppercase, Lowercase, Digit, Special Character</p>



	<p>Error Message: “Passwords must be at least 8 characters and contain at 3 of 4 of the following: upper case (A-Z), lower case (a-z), number (0-9) and special character (e.g. !@#\$%^&amp;*)”</p> <p>Motorosit application invalid if Password does not match Confi Password</p> <p>Error message: “Password does not match”</p>
Exceptions:	N. A
Includes:	N. A
Special Requirements:	N. A
Assumptions:	N. A
Notes and Issues:	N. A

Use Case ID:	002		
Use Case Name:	<b>Manage Profile</b>		
Created By:	Jia Wei	Last Updated By:	Jun Le
Date Created:	21/8/2019	Date Last Updated:	9/11/2019

Actor:	Motorist
Description:	Motorist is able to update their information or delete their account
Preconditions:	Motorist have logged in the application

Postconditions:	Motorist have updated their information or deleted their account
Priority:	Medium
Frequency of Use:	Motorist will use this when they wish to change their profile settings
Flow of Events:	<ol style="list-style-type: none"> <li>1. Motorist logged into the application</li> <li>2. Click on Profile</li> <li>3. Once Motorist is in the Profile Page: <ol style="list-style-type: none"> <li>3.1. Motorist able to update their First Name, Last, Email Address, Phone Number</li> <li>3.2. Motorist able to change their password</li> </ol> </li> </ol>
Alternative Flows:	<p>Motorist enters First Name/Last Name less than 2 characters Error Message: "First Name/Last Name must be at least 2 characters long "</p> <p>Motorist forgot to add '@' under email: Error Message: "Please include an '@' in the email address. '(email)' is missing an '@'."</p> <p>Motorist key in number length less than 8 for phone number: Error Message: "Please match the requested format."</p>
Exceptions:	N.A
Includes:	Logged into the application
Special Requirements:	N. A
Assumptions:	N. A
Notes and Issues:	N. A

Use Case ID:	003		
Use Case Name:	<b>Manage Favorites</b>		
Created By:	Andrew	Last Updated By:	Nurul Sabrina
Date Created:	21/8/2019	Date Last Updated:	9/11/2019

Actor:	Motorist
Description:	User can add or delete a carpark on his favorites' list
Preconditions:	1. The motorist must be logged in to the application
Postconditions:	1. The motorist must be able to save or delete his desired carpark into his favorites
Priority:	Medium
Frequency of Use:	Often
Flow of Events:	<ol style="list-style-type: none"> <li>1. The motorist logs into the application</li> <li>2. The motorist clicks on the 'heart icon' on a carpark to add to favorites</li> <li>3. The motorist clicks on 'Favorites' to view his saved carparks</li> <li>4. The motorist is able to choose to delete carpark by clicking the bin icon</li> </ol>
Alternative Flows:	N.A.
Exceptions:	N.A.
Includes:	Login
Special Requirements:	N.A.

Assumptions:	N.A.
Notes and Issues:	N.A.

Use Case ID:	004		
Use Case Name:	<b>Login</b>		
Created By:	Nurul Sabrina	Last Updated By:	Nurul Sabrina
Date Created:	21/8/2019	Date Last Updated:	2/11/2019

Actor:	Motorist
Description:	Motorist logs into the application
Preconditions:	Motorist must have signed up for an account
Postconditions:	Motorist stays logged in the application
Priority:	High
Frequency of Use:	Whenever motorist wants to use the application, he has to login
Flow of Events:	<ol style="list-style-type: none"> <li>1. Motorist enter login page</li> <li>2. Motorist enters username</li> <li>3. Motorist enters password</li> <li>4. Motorist successfully logs in</li> </ol>
Alternative Flows:	<p>If the motorist enters wrong username, display message "The username/ password is incorrect"</p> <p>If motorist enters wrong password, display message "The username/ password is incorrect"</p>

	If the user forgets his password, he can choose to reset his password. A reset password link will be sent to your email.
Exceptions:	N.A.
Includes:	Sign Up
Special Requirements:	N. A
Assumptions:	N.A.
Notes and Issues:	N. A

Use Case ID:	005		
Use Case Name:	<b>View Carparks on Maps</b>		
Created By:	Jun Le	Last Updated By:	Jun Le
Date Created:	21/8/2019	Date Last Updated:	9/11/2019

Actor:	Motorist
Description:	The motorist views carpark markers on the maps
Preconditions:	The motorist must be logged in to the application
Postconditions:	The motorist must be able to view carpark markers on this maps
Priority:	Medium
Frequency of Use:	Frequent
Flow of Events:	1.The motorist logs into the application 2. The motorist goes to the Maps page

	<p>3. The motorist clicks on a car park and view car park information like real-time availability all categorized in vehicles type Car, Motorcycle, Heavy Vehicle</p> <p>4. The motorist clicks on Directions to show the route</p> <p>5. The motorist clicks on Show Route to get specific directions</p>
Alternative Flows:	N. A
Exceptions:	N. A
Includes:	Login
Special Requirements:	N. A
Assumptions:	N. A
Notes and Issues:	N.A.

Use Case ID:	006		
Use Case Name:	<b>View Carparks on Homepage</b>		
Created By:	Jun Le	Last Updated By:	Jun Le
Date Created:	21/8/2019	Date Last Updated:	9/11/2019

Actor:	Motorist
Description:	The motorist views a list of caparks and its information on homepage
Preconditions:	The motorist must be logged in to the application

Postconditions:	The motorist must be able to view a list of carparks in the homepage
Priority:	Medium
Frequency of Use:	Frequent
Flow of Events:	<ol style="list-style-type: none"> <li>1.The motorist logs into the application</li> <li>2.The motorist goes to the Home page</li> <li>3.The motorist clicks on a car park and view car park information like real-time availability and parking rates all categorized into vehicles type Car, Motorcycle, Heavy Vehicle</li> </ol>
Alternative Flows:	N. A
Exceptions:	N. A
Includes:	Login
Special Requirements:	N. A
Assumptions:	N. A
Notes and Issues:	N.A.

Use Case ID:	007		
Use Case Name:	<b>Calculate Parking Cost</b>		
Created By:	Jun Le	Last Updated By:	Jun Le

Date Created:	21/8/2019	Date Last Updated:	9/11/2019
---------------	-----------	--------------------	-----------

Actor:	Motorist
Description:	The motorist views a list of caparks and its information on homepage
Preconditions:	The motorist must be logged in to the application
Postconditions:	The motorist must be able to view a list of carparks in the homepage
Priority:	Medium
Frequency of Use:	Frequent
Flow of Events:	<ol style="list-style-type: none"> <li>1.The motorist logs into the application</li> <li>2.The motorist goes to the Home page</li> <li>3.The motorist clicks on a car park and view car park information like real-time availability and parking rates all categorized into vehicles type Car, Motorcycle, Heavy Vehicle</li> <li>4.The motorist can choose to calculate his parking cost by entering Start Date/Time, End Date/Time</li> </ol>
Alternative Flows:	N. A
Exceptions:	N. A
Includes:	Login
Special Requirements:	N. A
Assumptions:	N. A
Notes and Issues:	N.A.



Use Case ID:	008		
Use Case Name:	<b>Search Carparks</b>		
Created By:	Jun Le	Last Updated By:	Nurul Sabrina
Date Created:	21/8/2019	Date Last Updated:	2/11/2019

Actor:	Motorist
Description:	The motorist searches for the desired carparks by carpark name or street name, and a list of carparks is returned
Preconditions:	<ol style="list-style-type: none"> <li>1. The motorist have logged into the application</li> </ol>
Postconditions:	<ol style="list-style-type: none"> <li>1. The motorist receives a list of carparks on his phone</li> <li>2. The motorist receives a message that no carparks are found</li> </ol>
Priority:	High
Frequency of Use:	Often
Flow of Events:	<ol style="list-style-type: none"> <li>1. The motorist login into the application</li> <li>2. The motorist enters the carpark name or street name</li> <li>3. The motorist views a list of carparks available</li> </ol>
Alternative Flows:	N.A.
Exceptions:	N. A
Includes:	Login

Special Requirements:	N. A
Assumptions:	N. A
Notes and Issues:	N. A

Use Case ID:	009		
Use Case Name:	<b>Filter Carparks</b>		
Created By:	Emmi	Last Updated By:	Andrew
Date Created:	21/8/2019	Date Last Updated:	2/11/2019

Actor:	Motorist
Description:	The motorist can choose his filter options by Ascending Order, Electronic Parking, Agency Type, Vehicle Type and Central Area or Location/Range
Preconditions:	The motorist needs to be logged in
Postconditions:	The motorist must be able to filter his search with the desired parameters
Priority:	High
Frequency of Use:	Often
Flow of Events:	1. The motorist logs into the application

	<ol style="list-style-type: none"> <li>2. The motorist can choose his filter options by Ascending Order, Electronic Parking, Agency Type, Vehicle Type and Central Area or Location/Range</li> <li>3. The motorist presses “Filter” button</li> <li>4. The motorist views a list of filtered carparks</li> </ol>
Alternative Flows:	N. A
Exceptions:	N. A
Includes:	Login
Special Requirements:	N. A
Assumptions:	N. A
Notes and Issues:	N. A

Use Case ID:	010		
Use Case Name:	<b>Provide Feedback</b>		
Created By:	Chun Foong	Last Updated By:	Jun Le
Date Created:	21/8/2019	Date Last Updated:	9/11/2019

Actor:	Motorist
--------	----------

Description:	Motorist can provide feedback when they face any problems or have any suggestions for improvement.
Preconditions:	The motorist must be logged in
Postconditions:	Motorist must be able to successfully post feedback
Priority:	Low
Frequency of Use:	NA
Flow of Events:	<ol style="list-style-type: none"> <li>1. Motorist logs in to the app</li> <li>2. Motorist provides feedback by clicking the feedback button on the top right</li> <li>3. Motorist enters Topic and Description</li> </ol>
Alternative Flows:	NA
Exceptions:	NA
Includes:	Login
Special Requirements:	NA
Assumptions:	N.A.
Notes and Issues:	NA

Use Case ID:	011
Use Case Name:	<b>Manage Accounts</b>

Created By:	Jia Wei	Last Updated By:	Jun Le
Date Created:	1/9/2019	Date Last Updated:	9/11/2019

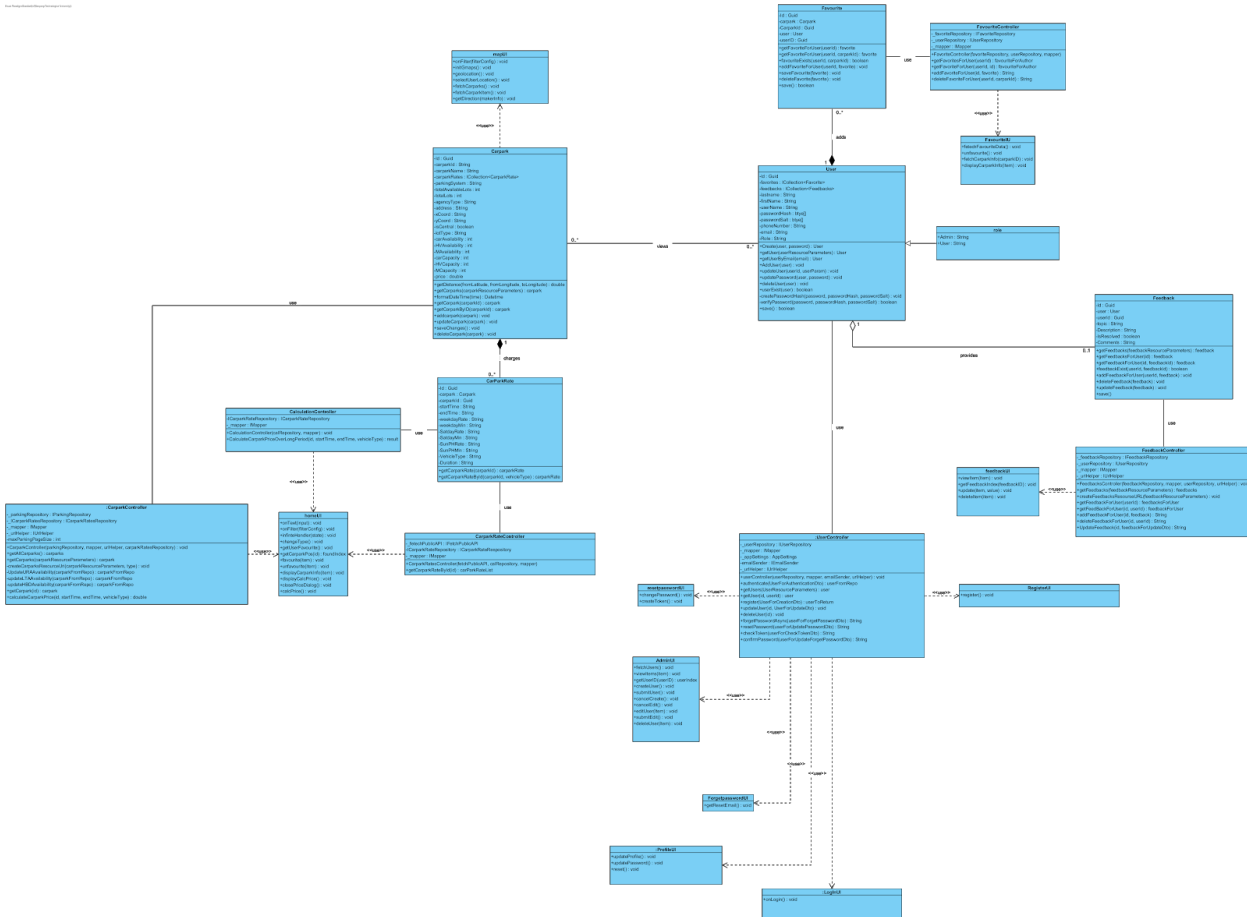
Actor:	Web App Administrator
Description:	The administrator can create, retrieve, update and delete user/admin accounts
Preconditions:	The web app administrator must be logged in
Postconditions:	Administrator must be able to successfully create, retrieve update and delete user/admin accounts
Priority:	High
Frequency of Use:	Low
Flow of Events:	<ol style="list-style-type: none"> <li>1. The web app administrator logs into the application</li> <li>2. The web app administrator clicks on Admin</li> <li>3. The web app administrator chooses to create, retrieve update and delete user/admin accounts</li> </ol>
Alternative Flows:	NA
Exceptions:	NA
Includes:	Login
Special Requirements:	NA
Assumptions:	N.A.
Notes and Issues:	NA

Use Case ID:	012		
Use Case Name:	<b>Manage User Feedbacks</b>		
Created By:	Andrew	Last Updated By:	Jun Le
Date Created:	1/9/2019	Date Last Updated:	9/11/2019

Actor:	Web App Administrator
Description:	The administrator can view,resolve,revoke and delete feedback
Preconditions:	The web app administrator must be logged in
Postconditions:	The administrator must be able to view,resolve,revoke and delete feedback successfully
Priority:	High
Frequency of Use:	High
Flow of Events:	<ol style="list-style-type: none"> <li>1. The web app administrator logs into the application</li> <li>2. The web app administrator clicks on Admin</li> <li>3. The web app administrator clicks on Manage Feedback</li> <li>4. The web app administrator deletes the errors after he has solved the errors</li> <li>5. The web app administrator can click View Feedback, and enter comments and resolve or revoke according his desired choice</li> <li>6. The web app administrator can click Delete Feedback remove</li> </ol>
Alternative Flows:	NA

Exceptions:	NA
Includes:	Login
Special Requirements:	NA
Assumptions:	N.A.
Notes and Issues:	NA

#### 4. Class Diagram

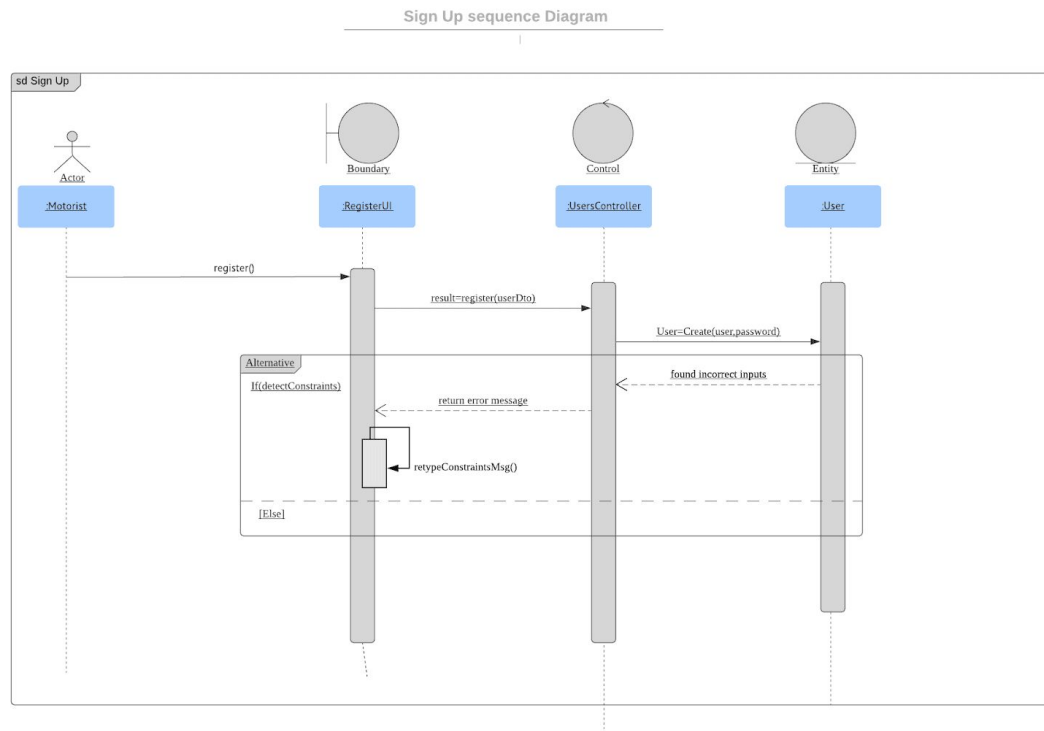


**Refer to file attached in SVN for better resolution.**

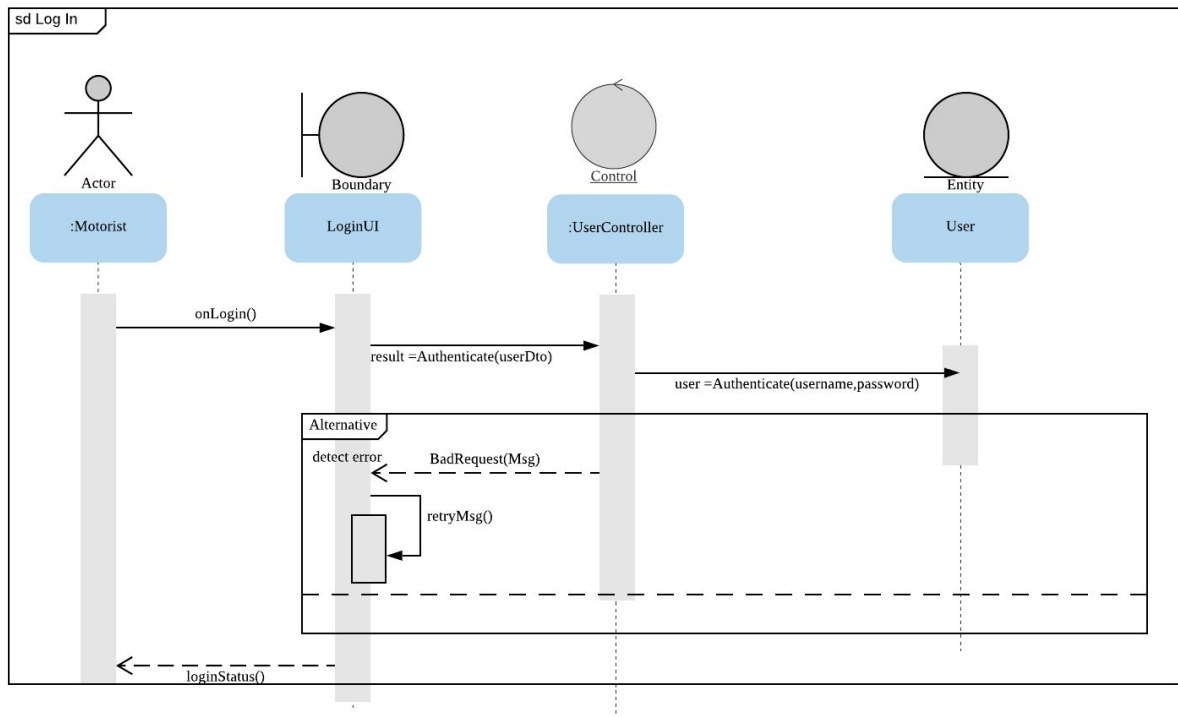


## 5. Sequence Diagram

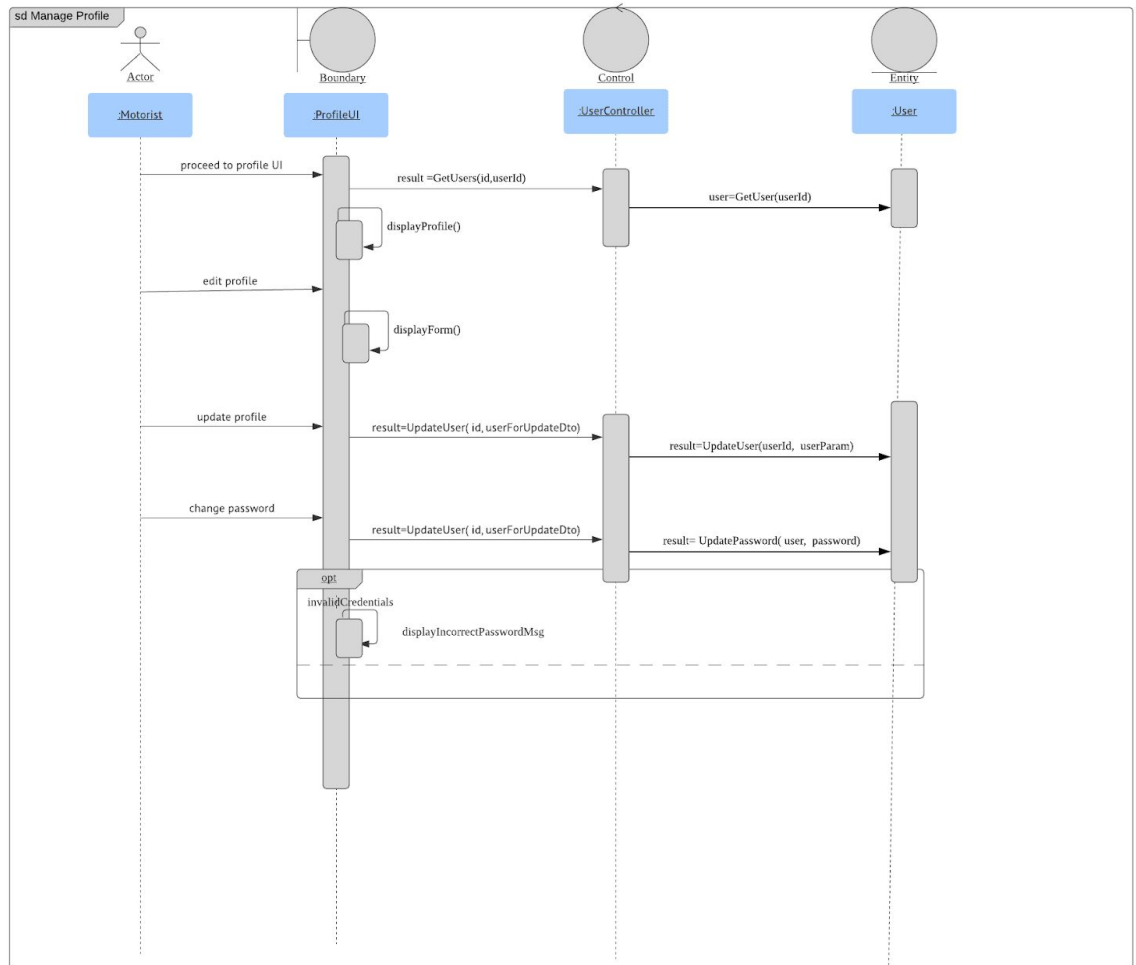
### 5.1. Sign up



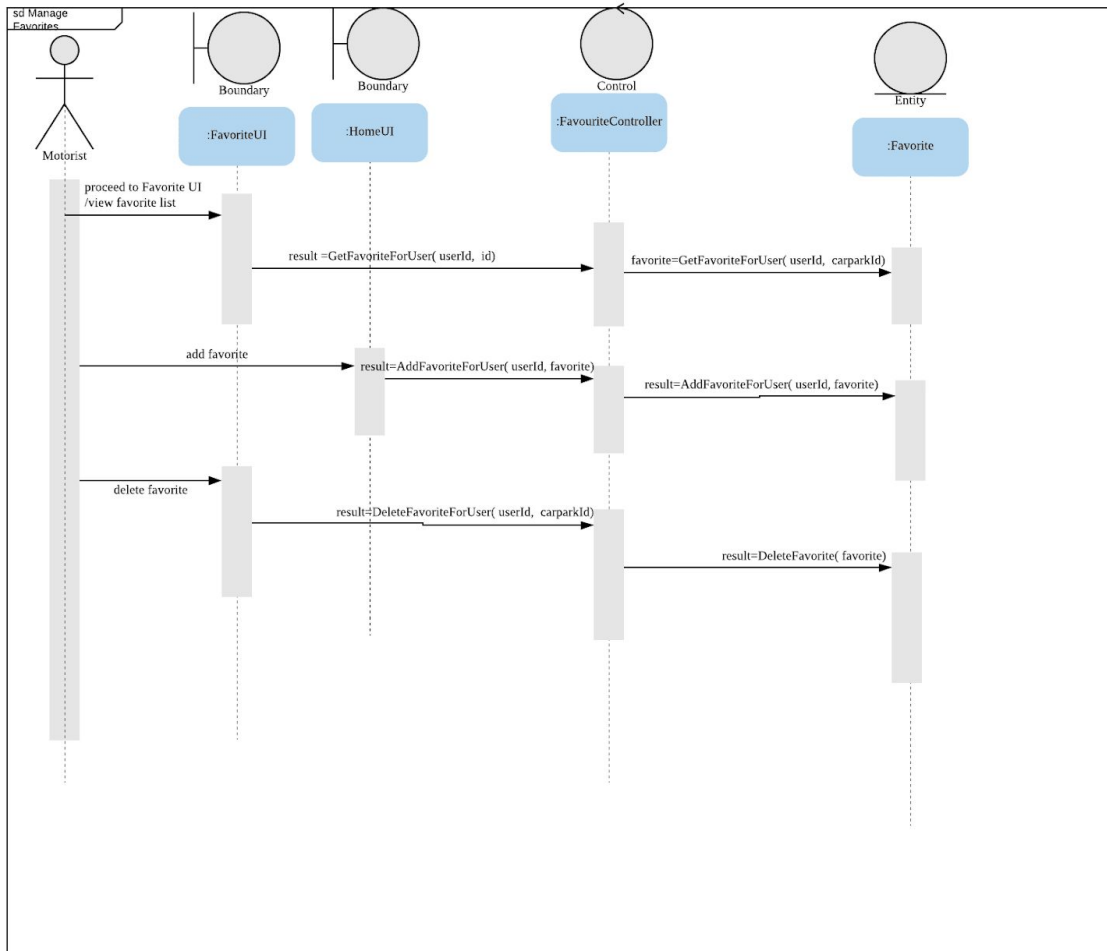
## 5.2. Login



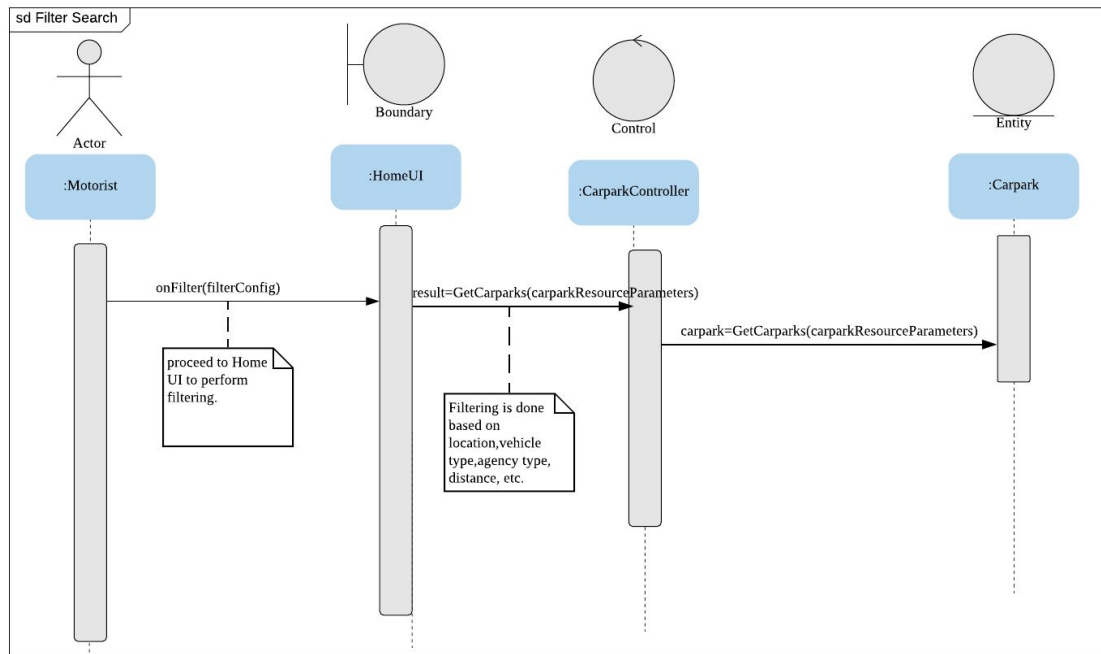
### 5.3. Manage Profile



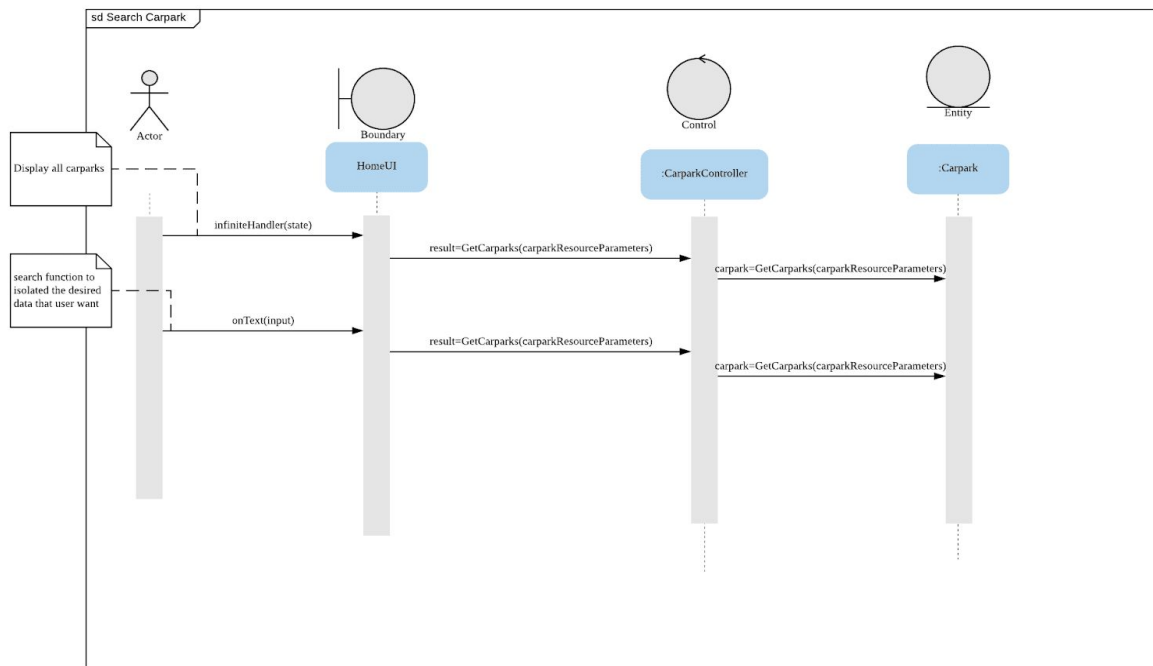
## 5.4. Manage Favorites



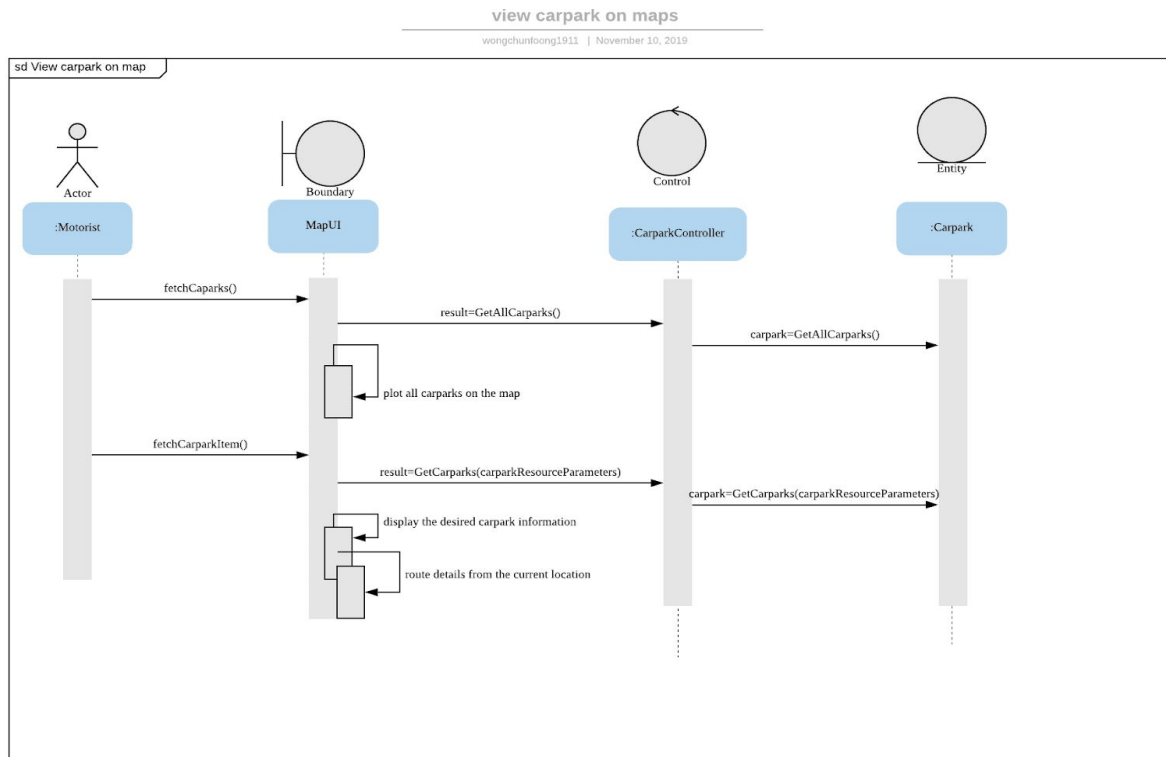
## 5.5. Filter Search



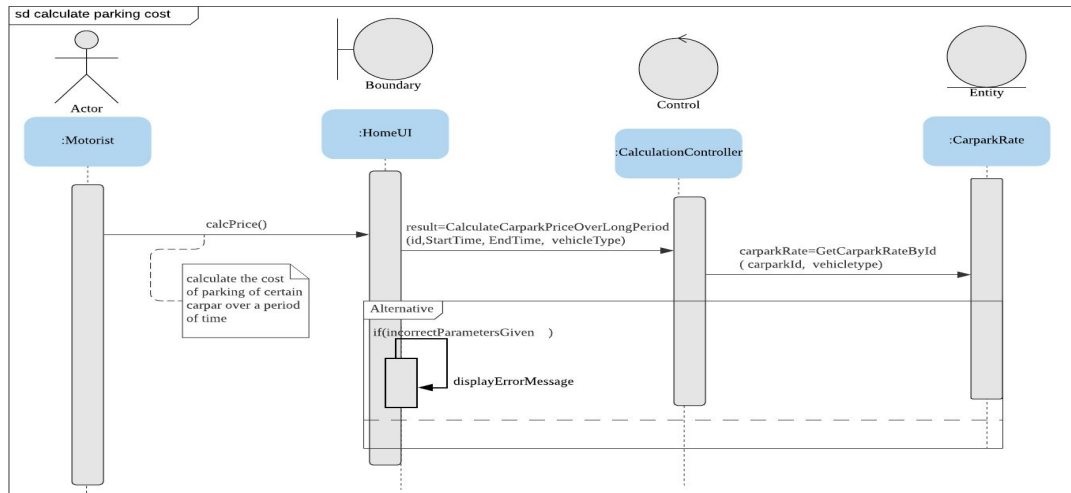
## 5.6. Search Carparks



## 5.7. View Carpark on Maps

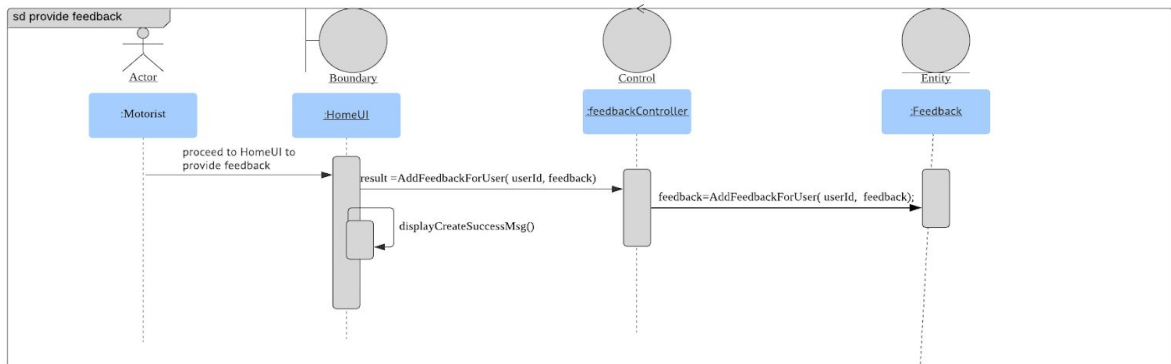


## 5.8. Calculate Parking Cost

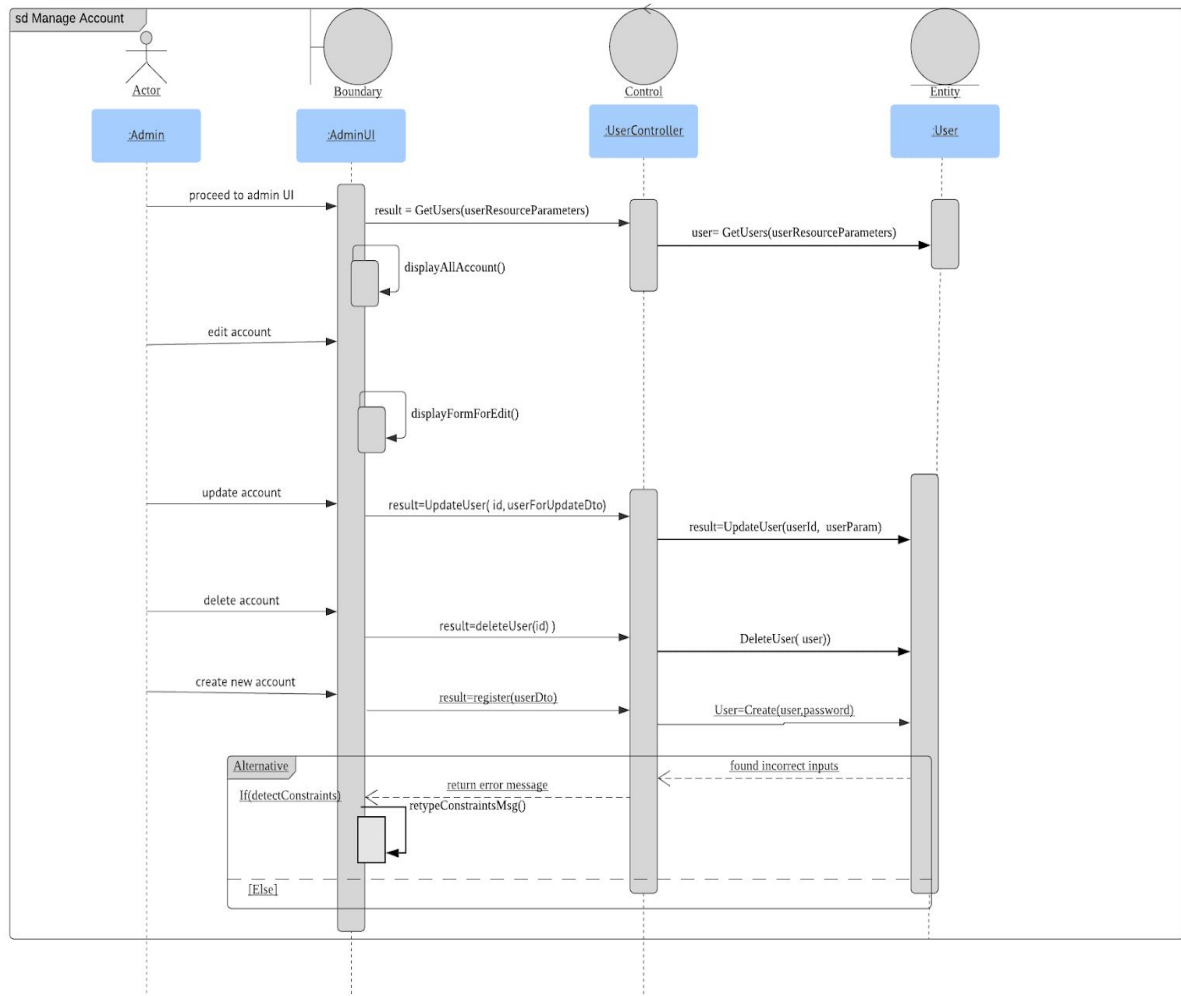




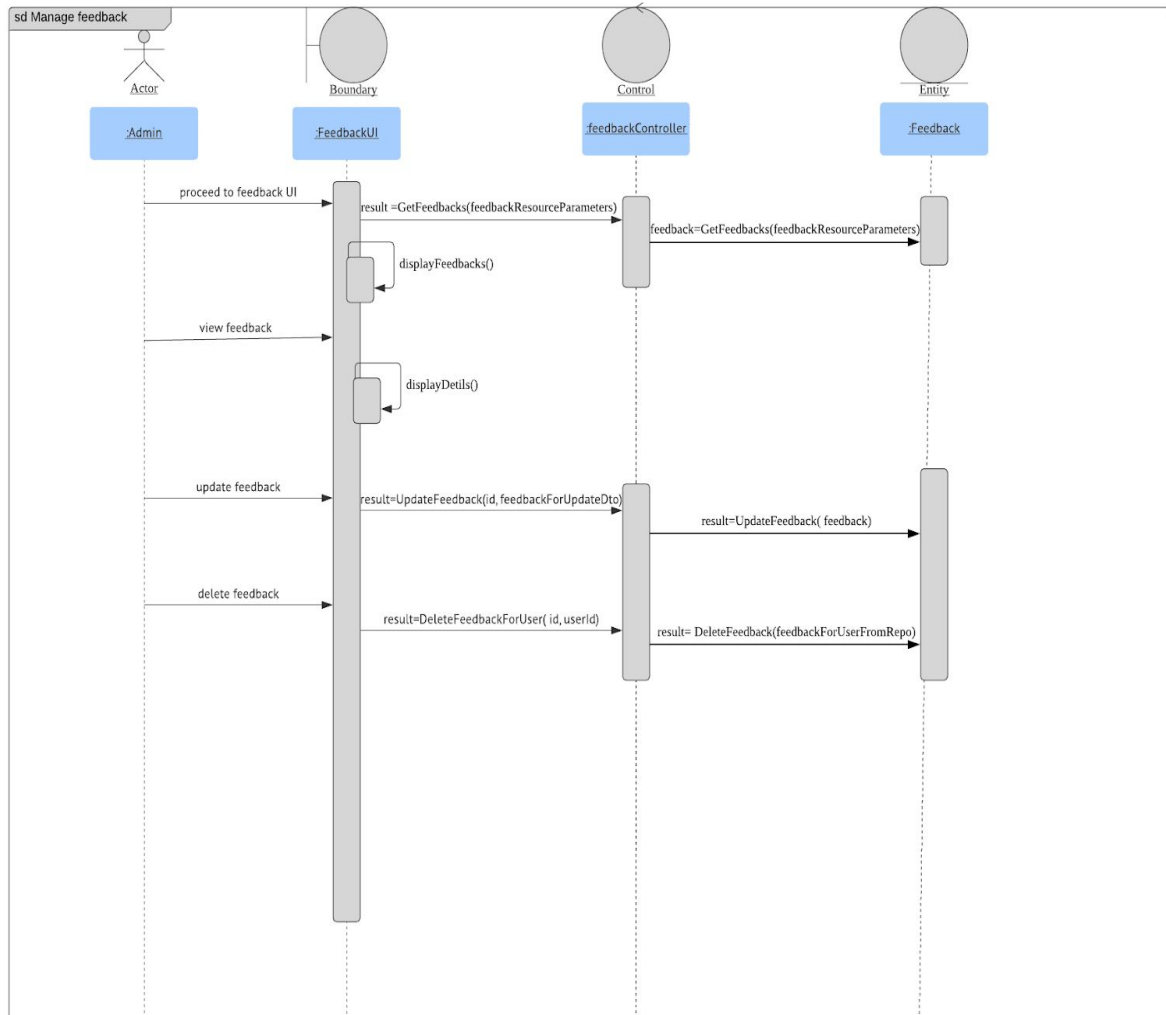
## 5.9. Provide Feedback



## 5.10. Manage User Accounts (Admin)

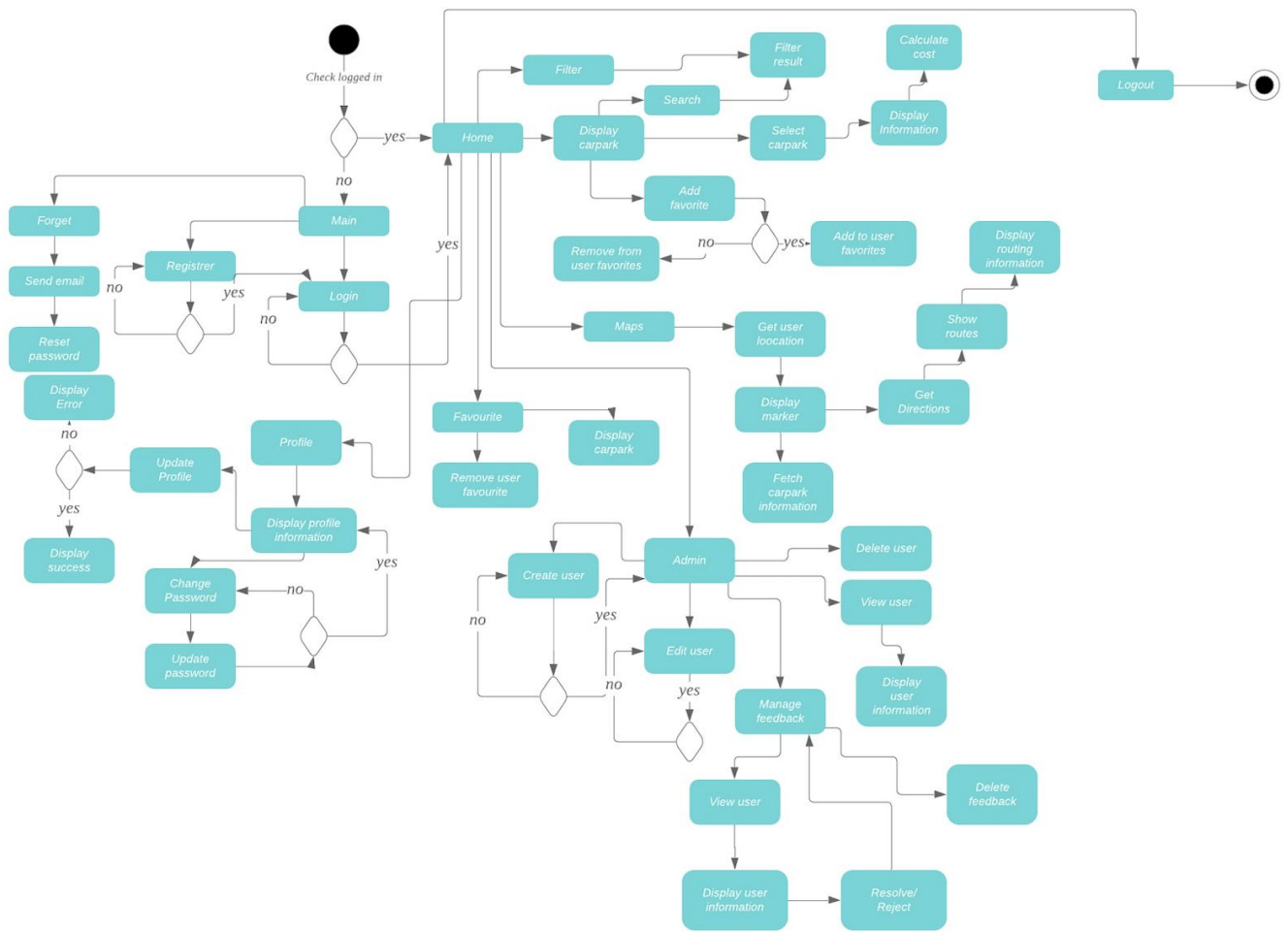


## 5.11. Manager User Feedbacks (Admin)

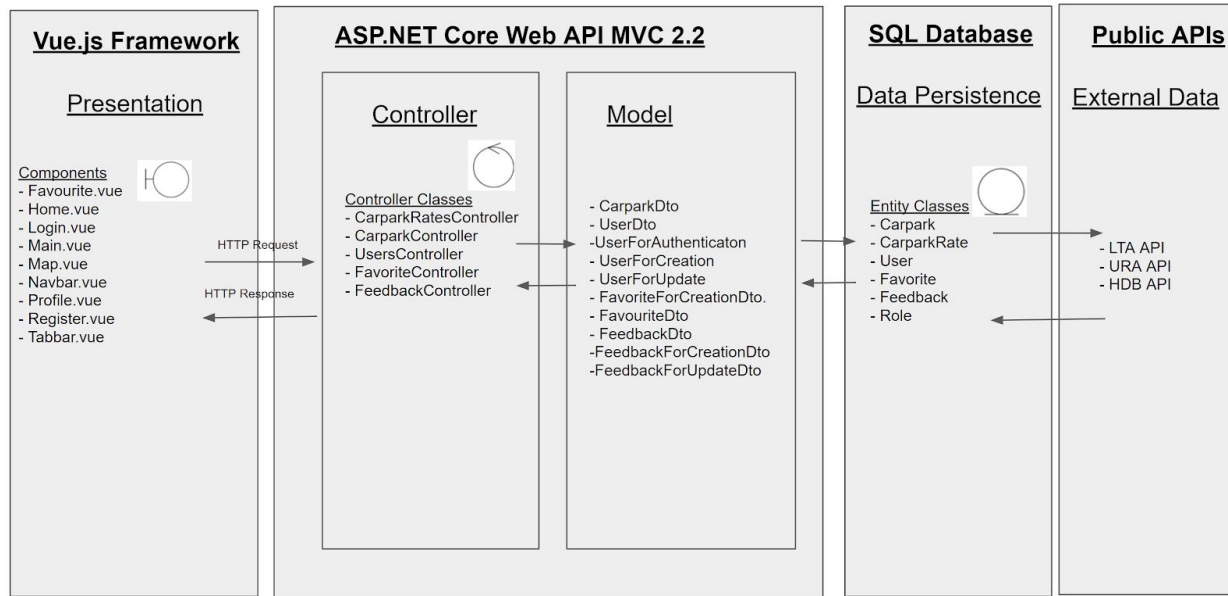




## 6. State Diagram



## 7. Application Architecture



### - Frontend Good Software Engineering Practices

- Decomposing VueJS components
  - By creating smaller components like navbar, tabbar and search filter which will be used in any other components
  - This makes code cleaner, easier to understand/maintain and in addition, it allows **reusability of the code** without having to repeat the same particular code over again. (DRY Principle)
- Consistent & Responsive design
  - Ensure that there's consistency in the layout, colours, fonts, forms and button to provide a better user experience. This is done by using a Material Design component framework called Vuetify.
  - Provides different views of the website through Mobile, tablet and desktop
- Improving page speed
  - Lazy loading components so that we will only fetch what we need - In case if the Javascript bundle gets too large and thus affecting the page load time

- Applies pagination wherever it is required - carpark data, user list, feedback so as to ensure **scalability**
- **Separation of concerns**
  - Admin folder - components whereby only an admin will be able to access
  - Auth folder - containing components where the user/admin authenticate with the backend
  - Layout folder - containing tabbar, navbar custom components
  - Plugin folder - containing library plugins
  - Store folder - state management setup which contain a config file and a modules folder to store modules
- User Experience
  - By implementing validation, error messages, feedback, it helps with improving the user experience as there might be times whereby the user inputs a wrong field which through validation the user will then be able to resolve it. Same goes with error messages
- **Backend Good Software Engineering Practices**
  - The backend is comprised of three components: **Public API, database and the Private API built by us**. Private API is responsible for populating data to database from Public API, serialisation of data as a result of CRUD operations, and generation of a payload response for the client.
  - Firstly, data from public API was called. Resulting rubbish data would be cleaned out, leaving valuable data stored in the database, such that it would be more easily accessible for private API to do CRUD operations tailored to the client.
  - Private API is built by the ASP.NET Core Web API framework, inspired by the **Model-View-Controller pattern**. Endpoint calls made by the client represent the view of the application, and controllers consists of methods that listen to the endpoint calls. Controllers handle the request by calling the database layer to perform CRUD operations. The database layer is represented by entity classes that form the model.
  - With the database populated, CRUD operations would need to be performed. The Entity Framework Core library was used, which implements Object Relational Mapping techniques. Database tables and columns could now be mapped into C# objects represented by the Model in MVC. This reduces

programming complexity as data manipulation would be done via LINQ queries rather than SQL.

- Controller classes were constructed with methods that would listen to the view, which are essentially HTTP endpoint calls made by the client. Controller methods call the database layer using constructor dependency injection techniques, abstracting the database and controller layer, making it loosely coupled.
- To construct the payload response performed by the API, the Data Access Object pattern was used. This pattern **isolates the application/business layer from the persistence layer**, represented by the entity database Model class. Essentially the controller would spit out Data Access objects that were mapped using the AutoMapper library from the Entity class. These objects would be transformed into a JSON payload response for the client to use.



## **8 Testing**

### **8.1 Blackbox Testing**

#### **Login**

Test ID	Description	Expected Result	Actual Result
1	1. Key in correct username 2. Key in correct password	Login into the application	Pass
2	1. Key in correct username 2. Key in incorrect password	Unsuccessful login.  Display "The username/password is incorrect"	Pass
3	1. Key in incorrect username 2. Key in correct password	Unsuccessful login.  Display "The username/password is incorrect"	Pass

## **Register Account**

Test ID	Description	Expected Result	Actual Result
1	<p>Precondition: user click “Register” button</p> <ol style="list-style-type: none"><li>1. Key in First Name</li><li>2. Key in Last Name</li><li>3. Key in Username</li><li>4. Key in Email</li><li>5. Key in phone number</li><li>6. Key in Password</li><li>7. Key in Confirm Password</li><li>8. Click “Sign Up” button</li></ol>	Successful account registration. User will be on Login Page.	Pass

2	<p>Precondition: user click "Register" button</p> <p>Precondition: existed username</p> <ol style="list-style-type: none"> <li>1. Key in First Name</li> <li>2. Key in Last Name</li> <li>3. Key in existed Username</li> <li>4. Key in Email</li> <li>5. Key in phone number</li> <li>6. Key in Password</li> <li>7. Key in Confirm Password</li> <li>8. Click "Sign Up" button</li> </ol>	<p>Unsuccessful account registration.</p> <p>Display " Username "(username)" is already taken"</p>	Pass
3	<p>Precondition: user click "Register" button</p> <p>Precondition: existed email</p> <ol style="list-style-type: none"> <li>1. Key in First Name</li> <li>2. Key in Last Name</li> <li>3. Key in Username</li> <li>4. Key in existed Email</li> <li>5. Key in phone number</li> <li>6. Key in Password</li> </ol>	<p>Unsuccessful account registration.</p> <p>Display "Email "(email)" is already taken"</p>	Pass

	7. Key in Confirm Password  8. Click "Sign Up" button		
4	Precondition: user click "Register" button  1. Key in First Name  2. Key in Last Name  3. Key in Username  4. Key in Email without '@'  5. Key in phone number  6. Key in Password  7. Key in Confirm Password  8. Click "Sign Up" button	Unsuccessful account registration.          Display "Please include a '@' in the email address. '(email)' is missing an '@'	Pass

5	<p>Precondition: user click "Register" button</p> <p>Precondition: existed phone number</p> <ol style="list-style-type: none"> <li>1. Key in First Name</li> <li>2. Key in Last Name</li> <li>3. Key in Username</li> <li>4. Key in Email</li> <li>5. Key in phone number</li> <li>6. Key in Password</li> <li>7. Key in Confirm Password</li> <li>8. Click "Sign Up" button</li> </ol>	<p>Unsuccessful account registration.</p> <p>Display "Phone number "(phone number)" is already taken"</p>	Pass
6	<p>Precondition: user click "Register" button</p> <ol style="list-style-type: none"> <li>1. Key in First Name</li> <li>2. Key in Last Name</li> <li>3. Key in Username</li> <li>4. Key in Email</li> <li>5. Key in phone number with length less than 8</li> <li>6. Key in Password</li> </ol>	<p>Unsuccessful account registration.</p> <p>Display "Please match the requested format."</p>	Pass

	<p>7. Key in Confirm Password</p> <p>8. Click “Sign Up” button</p>		
7	<p>Precondition: user click “Register” button</p> <p>1. Key in First Name</p> <p>2. Key in Last Name</p> <p>3. Key in Username</p> <p>4. Key in existed Email</p> <p>5. Key in phone number</p> <p>6. Key in Password</p> <p>7. Key in Confirm Password with different characters than previous password</p> <p>8. Click “Sign Up” button</p>	<p>Unsuccessful account registration.</p> <p>Display “Password does not match”</p>	

8	<p>Precondition: user click “Register” button</p> <ol style="list-style-type: none"> <li>1. Key in First Name</li> <li>2. Key in Last Name</li> <li>3. Key in Username</li> <li>4. Key in existed Email</li> <li>5. Key in phone number</li> <li>6. Key in Password with length less than 8/ no uppercase/no lower case/ no special characters/ no digit</li> <li>7. Key in Confirm Password</li> <li>8. Click “Sign Up” button</li> </ol>	<p>Unsuccessful account registration.</p> <p>Display “Passwords must be at least 8 characters and contain at 3 of 4 of the following : upper case (A-Z), lower case (a-z), number (0-9) and special character (e.g. !@#\$%^&amp;*)”</p>	
---	--	---	--

### **Search carpark**

Test ID	Description	Expected Result	Actual Result
1	1. Key in carpark	Show the list of carpark based on the user key in.	Pass
2	2. Key in invalid carpark name	Show the list with the similar carpark name user type in	Pass

### **Favorites**

Test ID	Description	Expected Result	Actual Result
1	1. Click bin icon 2. Message prompt "Are you sure" 3. Click "Ok" button	The carpark will be deleted from favorites	Pass
2	1. Click bin icon 2. Message prompt "Are you sure" 3. Click "Cancel" button	The carpark will be not be deleted from favorites	Pass



### **Focus user to current location**

Test ID	Description	Expected Result	Actual Result
1	1. Click on the target icon	The current location will be detected	Pass

### **Filtering of carparks**

Test ID	Description	Expected Result	Actual Result
1	1. Enable Ascending Order 2. Enable Electronic Parking 3. Disable central area 4. Type in the location 5. Select Car Type 6. Select Agency Type 7. Type in the range	Show list of carparks within the specific location and range.	Pass

2	<ol style="list-style-type: none"> <li>1. Enable Ascending Order</li> <li>2. Enable Electronic Parking</li> <li>3. Enable central area</li> <li>4. Location is not set</li> <li>5. Select Car Type</li> <li>6. Select Agency Type</li> <li>7. Type in the range</li> </ol>	Show list of carparks from the central area and other filtered variables.	Pass
3	<ol style="list-style-type: none"> <li>1. Enable Ascending Order</li> <li>2. Disable Electronic Parking</li> <li>3. Disable central area</li> <li>4. Location is not set</li> <li>5. Car Type not set</li> <li>6. Agency Type not set</li> <li>7. Range that is set by default (1000m)</li> </ol>	Show list of carparks that are not electronic parking in an ascending order.	Pass

4	<ol style="list-style-type: none"> <li>1. Enable Ascending Order</li> <li>2. Enable Electronic Parking</li> <li>3. Enable Central Area</li> <li>4. Location is not set</li> <li>5. Choose Car Type (3 options: Car, Motorcycle or Heavy Vehicles)</li> <li>6. Choose Agency type (LTA or URA)</li> <li>7. Range that is set by default (1000m)</li> </ol>	<p>Show no carparks</p> <p>Display “—No Carpark--”</p>	Pass
---	---	--	------

### Change Password

Test ID	Description	Expected Result	Actual Result
1	<ol style="list-style-type: none"><li>1. Key in current password</li><li>2. Key in new password</li><li>3. Key in confirm new password</li><li>4. Click "Update Password" button</li></ol>	<p>Successful change of password</p> <p>Display "Password has been successfully updated!"</p>	Pass
2	<ol style="list-style-type: none"><li>1. Key in current username</li><li>2. Key in new password</li><li>3. Key in confirm new password with different characters from new password</li><li>4. Click "Update Password" button</li></ol>	<p>Unsuccessful change of password.</p> <p>Display "New password does not match"</p>	Pass
3	<ol style="list-style-type: none"><li>1. Key in current username</li><li>2. Key in new password with length less than 8</li><li>3. Key in confirm new password</li><li>4. Click "Update Password" button</li></ol>	<p>Unsuccessful change of password</p> <p>Display "Passwords must be at least 8 characters and contain at 3 of 4 of the following: upper case (A-Z), lower case (a-z), number (0-9), and special character (!@#\$%^&amp;*~)."</p>	Pass

		and special character (e.g. !@#\$%^&*)"	
4	<ol style="list-style-type: none"> <li>1. Key in current username</li> <li>2. Key in new password with now upper case</li> <li>3. Key in confirm new password</li> <li>4. Click "Update Password" button</li> </ol>	<p>Unsuccessful change of password</p> <p>Display "Passwords must be at least 8 characters and contain at 3 of 4 of the following upper case (A-Z), lower case (a-z), number (0-9) and special character (e.g. !@#\$%^&amp;*)"</p>	Pass
	<ol style="list-style-type: none"> <li>1. Key in current username</li> <li>2. Key in new password with no lower case</li> <li>3. Key in confirm new password</li> <li>4. Click "Update Password" button</li> </ol>	<p>Unsuccessful change of password</p> <p>Display "Passwords must be at least 8 characters and contain at 3 of 4 of the following upper case (A-Z), lower case (a-z), number (0-9) and special character (e.g. !@#\$%^&amp;*)"</p>	Pass

	<ol style="list-style-type: none"> <li>1. Key in current username</li> <li>2. Key in new password with no special character</li> <li>3. Key in confirm new password</li> <li>4. Click "Update Password" button</li> </ol>	<p>Unsuccessful change of password</p> <p>Display "Passwords must be at least 8 characters and contain at 3 of 4 of the following: upper case (A-Z), lower case (a-z), number (0-9) and special character (e.g. !@#\$%^&amp;*)"</p>	Pass
	<ol style="list-style-type: none"> <li>1. Key in current username</li> <li>2. Key in new password with no digit</li> <li>3. Key in confirm new password</li> <li>4. Click "Update Password" button</li> </ol>	<p>Unsuccessful change of password</p> <p>Display "Passwords must be at least 8 characters and contain at 3 of 4 of the following: upper case (A-Z), lower case (a-z), number (0-9) and special character (e.g. !@#\$%^&amp;*)"</p>	Pass

## **Manage Profile**

Test ID	Description	Expected Result	Actual Result
1	<ol style="list-style-type: none"><li>1. Key in First Name</li><li>2. Key in Last Name</li><li>3. Key in Email Address</li><li>4. Key in Contact Number</li><li>5. Click "Save Changes" button</li></ol>	<p>Successful update of profile</p> <p>Display "Profile has been successfully updated"</p>	Pass
2	<ol style="list-style-type: none"><li>1. Key in First Name</li><li>2. Key in Last Name</li><li>3. Key in Email Address without '@'</li><li>4. Key in Contact Number</li><li>5. Click "Save Changes" button</li></ol>	<p>Unsuccessful profile update.</p> <p>Display "Please include an '@' in the email address. '(email)' is missing an '@'."</p>	Pass
3	<ol style="list-style-type: none"><li>1. Key in First Name</li><li>2. Key in Last Name</li><li>3. Key in Email Address</li><li>4. Key in Contact Number less than 8 digits</li><li>5. Click "Save Changes" button</li></ol>	<p>Unsuccessful account registration.</p> <p>Display "Please match the requested format."</p>	Pass

## **Reset Password**

Test ID	Description	Expected Result	Actual Result
1	<ol style="list-style-type: none"><li>1. Enter email</li><li>2. Click “Reset” button</li></ol>	<p>Successfully sent reset password to motorist's email address</p> <p>Display “Check your email (user email address) to reset your password!”</p>	Pass
2	<ol style="list-style-type: none"><li>1. Enter email without '@'</li><li>2. Click “Reset” button</li></ol>	<p>Unsuccessful to send reset password.</p> <p>Display “Please include an '@' in the email address. '(email)' is missing an '@'.”</p>	Pass



## **Feedback**

Test ID	Description	Expected Result	Actual Result
1	<ol style="list-style-type: none"><li>1. Type in the topic of the feedback</li><li>2. Type in the description</li><li>3. Click save</li></ol>	<p>Feedback is successfully created</p> <p>Display "Feedback created"</p>	Pass

### Calculate Price

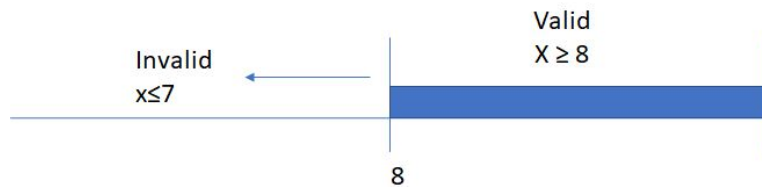
Test ID	Description	Expected Result	Actual Result
1	<ol style="list-style-type: none"><li>1. Select Start Date/Time</li><li>2. Select End Date/Time</li><li>3. Select Vehicle Type: Car</li><li>4. Click "Calculate" button</li></ol>	<p>Carpark price is shown</p> <p>Display "Carpark price \$XX.XX"</p>	Pass
2	<ol style="list-style-type: none"><li>1. Start Date/Time not selected</li><li>2. Select End Date/Time</li><li>3. Select Vehicle Type: Car</li><li>4. Click "Calculate" button</li></ol>	<p>Carpark price shown is not successful</p> <p>Display "Pricing information for this carpark is unavailable this moment. Please try again in the future"</p>	Pass
3	<ol style="list-style-type: none"><li>1. Select Start Date/Time</li><li>2. End Date/Time is not selected</li><li>3. Select Vehicle Type: Car</li><li>4. Click "Calculate" button</li></ol>	<p>Carpark price shown is not successful</p> <p>Display "Pricing information for this carpark is unavailable this moment. Please try again in the future"</p>	Pass

4	<ol style="list-style-type: none"> <li>1. Select Start Date/Time</li> <li>2. End Date/Time is not selected</li> <li>3. Select Vehicle Type is not selected</li> <li>4. Click "Calculate" button</li> </ol>	<p>Carpark price shown is not successful</p> <p>Display "Pricing information for this carpark is unavailable this moment. Please try again in the future"</p>	Pass
---	--	---	------

## Boundary Value Testing (Inputs with Range of Values)

### Test cases for Register Account

Length of password valid equivalence class ( $x \geq 8$ )



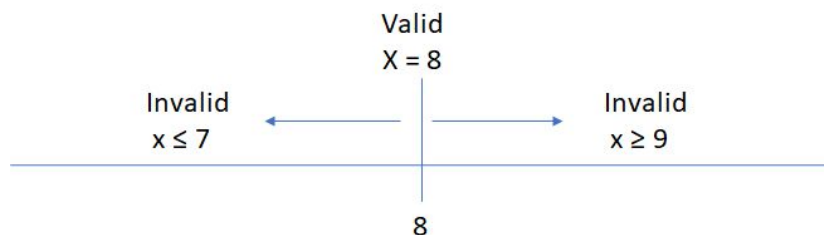
Invalid Equivalence Class ( $0 \leq x \leq 7$ )

- Lower boundary 0: 0, 1
- Upper Boundary 7: 6, 7, 8

Valid Equivalence Class ( $x \geq 8$ )

- Lower boundary 8: 7, 8, 9
- **Valid boundary values: {8}**
- **Invalid boundary values: {0, 7}**

Length of phone number valid equivalence class ( $x = 8$ )



Valid Equivalence Class ( $X = 8$ )

- Lower/Upper Boundary Boundary : 7,8,9

Invalid Equivalence Class (  $0 \leq X \leq 7$  )

- Lower boundary : 0,4
- Upper boundary: 6,7,8

Invalid Equivalence Class (  $X \geq 9$  )

- Lower boundary: -8,9,40
- Valid boundary values: {8}
- Invalid boundary values: {0,7,9}

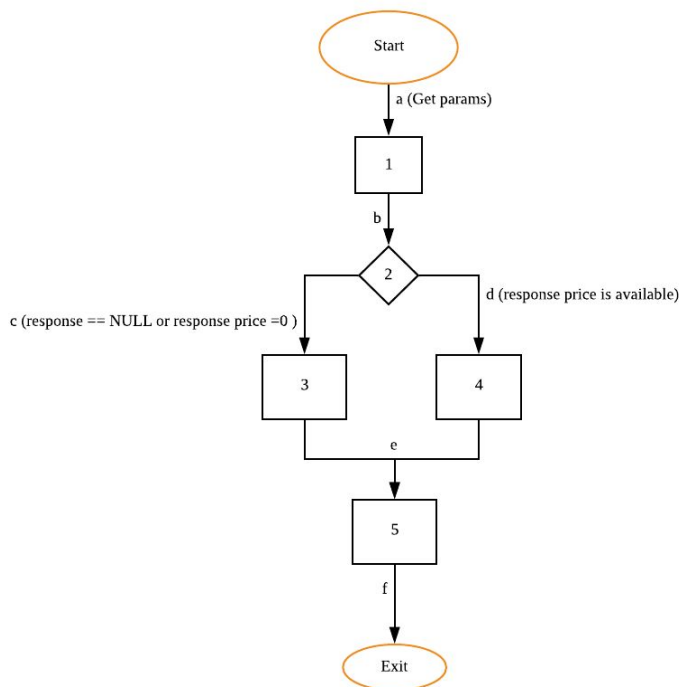
Password Input Length	Phone Number Input Length	Expected Output	Actual Output
8 (Valid)	8 (Valid)	Accept	Accept
8 (Valid)	0 (Invalid)	Reject	Reject
8 (Valid)	7 (Invalid)	Reject	Reject
8 (Valid)	9 (Invalid)	Reject	Reject
0 (Invalid)	8 (Valid)	Reject	Reject
0 (Invalid)	8 (Valid)	Reject	Reject

All boundary values test cases have passed for Register Account.

## 8.2 White Box Testing

### Calculate Price

```
calcPrice: function() {  
  let cur = this;  
  this.axios  
  .get(  
    "https://parkingslotapi.azurewebsites.net/api/calculation/" +  
    this.carparkPriceItem.id,  
    {  
      params: this.priceFilter ①  
    }  
  )  
  .then(response => { ②  
    if (response.data.isNull == true || response.data.price == 0) {  
      cur.alertPrice = true; ③  
      cur.successPrice = false;  
    } else {  
      cur.successPrice = true; ④  
      cur.alertPrice = false;  
      cur.calculatedPrice = response.data.price;  
    }  
    cur.$refs.form.reset(); ⑤  
  })  
  .catch(error => {  
    console.log(error);  
  });  
},  
closePriceDialog() {  
  this.$refs.form.reset();  
  this.costDialog = false;  
}
```



### Independent Test Paths

Path 1: a-b-c-e-f

Path 2: a-b-c-d-f

#### Path 1

Input valid parameters and the price is either not given (NULL) or price is \$0

Start Date/Time: 14:00; 9/11/2019, End Date/Time: 14:00; 10/11/2019, Vehicle Type: Car

Expected Result:

Display: Pricing information for this carpark is unavailable at this moment. Please try again in the future

#### Path 2

Input valid parameters and the price is available.

Start Date/Time: 14:00; 9/11/2019, End Date/Time: 14:00; 10/11/2019, Vehicle Type: Car

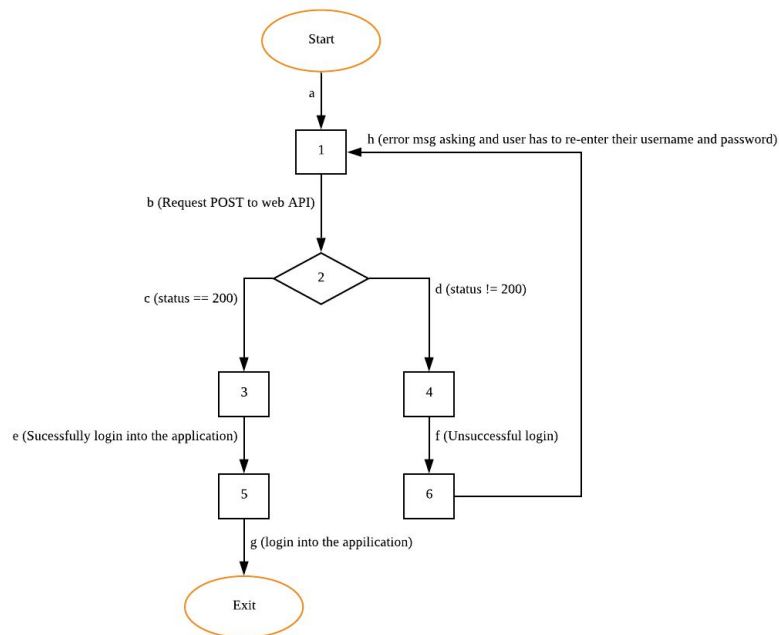
Expected Result:

Display: Carpark price is \$28.80

## Login

```
onLogin() {  
  /* Send a post req to the backend server */  
  /* Set localStorage loginStatus to true */  
  /* Redirect the user into the app */  
  this.$store  
    .dispatch("LOGIN", {  
      Username: this.username, ①  
      Password: this.password  
    })  
    .then(success => {  
      this.$router.push("/home"); ⑤  
    })  
    .catch(error => {  
      this.error = true; ⑥  
    });  
}
```

```
LOGIN: ({ commit }, payload) => {  
  return new Promise((resolve, reject) => { ②  
    /* POST to the Web API */  
    axios.post('https://parkingslotapi.azurewebsites.net/api/users/authenticate', payload).then(({ data, status }) => {  
      if (status === 200) {  
        //Add user info to store  
        store.commit('SETUSERNAME', data.username);  
        store.commit('SETEMAIL', data.email);  
        store.commit('SETFIRSTNAME', data.firstName);  
        store.commit('SETLASTNAME', data.lastName); ③  
        store.commit('SETPHONENO', data.phoneNumber);  
        store.commit('SETROLE', data.role);  
        store.commit('SETTOKEN', data.token);  
        store.commit('SETUSERID', data.id);  
        store.commit('SETAUTHSTATUS', true);  
        resolve(true);  
      }  
    }).catch(error => { ④  
      reject(error);  
    });  
  });  
},
```





## Independent Test Paths

Path 1: a-b-c-e-g

Path 2: a-b-d-f-h

### Path 1

Valid username and password where user able to login into the ParkingSlot successfully

Username: light@04, Password: Password123!

Expected Result:

User successfully login into the ParkingSlot.

### Path 2

Invalid username/password where user unable to login into ParkingSlot.

Username: light@04, Password: Password1234!!

Expected Result:

User unsuccessfully login into the application and retypes the username and password.

## Register

```

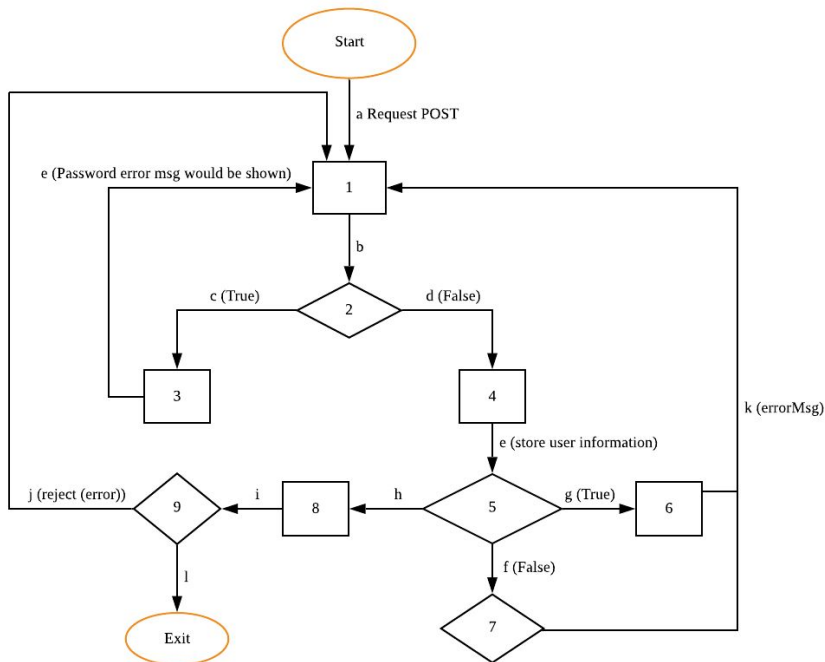
REGISTER: ({ commit }, payload) => {
  /* POST to the Web API */
  return new Promise((resolve, reject) => {
    axios
      .post("https://parkingslotapi.azurewebsites.net/api/users", payload)
      .then(({ data, status }) => {
        if (status === 200) {
          resolve(data);
        }
      })
      .catch(error => {
        //console.log(error.response.data);
        //console.log(error.response.status);
        reject(error);
      });
  });
};

```

```

register() {
  if (this.password !== this.confirmPassword) {
    this.passwordError = true;
    this.matchError = false;
    return;
  }
  this.passwordError = false;
  this.$store
    .dispatch("REGISTER", {
      FirstName: this.firstName,
      LastName: this.lastName,
      Username: this.username,
      Email: this.email,
      Role: "User",
      PhoneNumber: this.phoneNo,
      Password: this.password
    })
    .then(success => {
      this.$router.push("/login");
    })
    .catch(error => {
      console.log(error.response);
      console.log(error.response.data.message);
      if (error.response.status === 400) {
        if (error.response.data.message === undefined) {
          this.errorMsg = error.response.data.Password[0];
        } else {
          this.errorMsg = error.response.data.message;
        }
        this.matchError = true;
      } else {
        this.matchError = false;
        this.passwordError = false;
      }
    });
}

```



### Independent Test Paths

Path 1: a-b-c-e

Path 2: a-b-d-e-g-k

Path 3: a-b-d-e-f-k

Path 4: a-b-d-e-h-i-j

Path 5: a-b-d-e-h-i-l

#### Path 1

Motorist password and confirm password does not match.

Password: Password123!, Confirm Password: password123!

Expected Result:

Display "Password does not match"

#### Path 2 and Path 4

Motorist somehow unable to register upon registering with valid inputs

Expected Result:

Motorist will receive a message where they are not able to register.

#### Path 3

Motorist type in invalid/existed phone number, email, username

Username: light04, Phone Number: 12345678, Email: geh@gmail.com

Expected Result:

Display "Username "light04" is already taken"

Display "Phone number "12345678" is already taken"

Display "Email "geh@gmail.com" is already taken"

#### Path 5

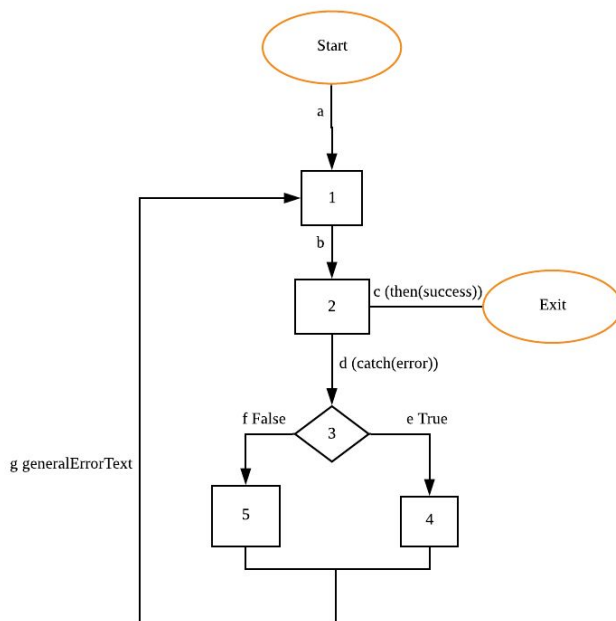
Motorist successfully registered their account

Expected Result:

Motorist will be directed to the login page

## Manage Profile

```
updateProfile() {  
  /* Axios PUT Req to update */  
  /* change state back to */  
  /* update vuex state for username */  
  this.$store  
    .dispatch("UPDATE", {  
      FirstName: this.userProfile.firstName, ①  
      LastName: this.userProfile.lastName,  
      Email: this.userProfile.Email,  
      PhoneNumber: this.userProfile.Contact  
    })  
    .then(success => {  
      this.generalError = false; ②  
      this.notifyStatus = true;  
      this.notifyText = "Profile have been successfully updated!";  
    })  
    .catch(error => {  
      this.notifyStatus = false; ③  
      if (this.userProfile.firstName.length <= 1) {  
        this.generalErrorText = error.response.data.FirstName[0];  
        this.generalError = true; ④  
      }  
      else if (this.userProfile.lastName.length <= 1) {  
        this.generalErrorText = error.response.data.LastName[0];  
        this.generalError = true; ⑤  
      }  
    });  
}
```



## Independent Test Paths

Path 1: a-b-c

Path 2: a-b-d-e-g

Path 3: a-b-d-f-g

#### Path 1

Motorist update their data successfully.

Expected Result:

Display "Profile have been successfully updated!"

#### Path 2

Motorist left their first name empty or only 1 letter therefore motorist unable to update profile.

Expected Result:

Display "First Name is required."

Display "First Name must be at least 2 characters long."

#### Path 3

Motorist left out their last name empty or only type in 1 letter therefore motorist unable to update profile.

Expected Result:

Display "Last Name is required."

Display "Last Name must be at least 2 characters long."